

UNIVERSIDAD DE CUENCA



FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA DE SISTEMAS

“CREACIÓN DE COMPONENTES PARA EL FRAMEWORK DE
GENERACIÓN DE RESOURCE DESCRIPTION FRAMEWORK (RDF)”

desde 1867

Tesis previa a la obtención del
título de Ingeniero de Sistemas

Autor:

Fabián Leonardo Peñaloza Marín.
Edison Freddy Santacruz Mora.

Director:

Ing. Víctor Hugo Saquicela Galarza, PhD.

Cuenca - Ecuador
2015

Resumen

El proceso de generación de Datos Abiertos Enlazados (Linked Open Data - LOD en ingles) es tedioso, esto debido a la necesidad de poseer conocimientos en diferentes áreas. Actualmente, existen muchas herramientas que permiten convertir los datos disponibles en Internet a formato RDF, pero ninguna provee una manera intuitiva, integrada y ágil de realizar este proceso. En esta tesis se presenta una plataforma para la generación de LOD, que cubre los pasos de una metodología utilizada por la comunidad científica. La unificación de metodología y plataforma se realiza mediante la creación de componentes dentro de la herramienta Pentaho Data Integration (PDI). Estos componentes permiten extraer datos desde repositorios OAI-PMH, cargar las ontologías, crear mapeo entre los datos y ontologías, generar RDF, publicar un SPARQL Endpoint y explotarlo.

Palabras Clave: Linked Open Data (LOD), RDF, Pentaho Data Integration (PDI).

Abstract

The generation of Linked Open Data (LOD) is an overwhelming process, since a vast knowledge in several fields is needed. Currently, there are several tools that allow converting Internet available data into the RDF format, however, none of them provides an intuitive, integrated or suitable way to perform the conversion process. This thesis work presents a LOD generation platform that involves the corresponding steps related to a well accepted scientific community methodology. The fusion of the proposed platform and such methodology is performed by creating components within Pentaho Data Integration (PDI), that allows: data extraction from OAI-PMH repositories, data ontology mapping, RDF generation and, SPARQL Endpoint publication and exploiting.

Keywords: Linked Open Data (LOD), RDF, Pentaho Data Integration (PDI).

Fabian Leonardo Peñaloza Marin
Edison Freddy Santacruz Mora

Índice General

Resumen	b
Abstract	c
Índice General	I
Índice de figuras.	V
Índice de tablas.	VIII
1. Introducción	1
2. Marco Teórico	3
2.1. Introducción	4
2.2. Web Semántica	4
2.2.1. ¿Qué es la Web Semántica?	4
2.2.2. ¿Para qué Sirve la Web Semántica?	4
2.2.3. ¿Cómo Funciona la Web Semántica?	5
2.2.4. Herramientas de Web Semántica	6
2.2.4.1. <i>Resource Description Framework</i> (RDF)	6
2.2.4.2. <i>Terse RDF Triple Language</i> (Turtle)	7
2.2.4.3. <i>Query Language for RDF</i> (SPARQL)	8
2.2.4.4. <i>Web Ontology Language</i> (OWL)	8
2.3. Metodología para la Publicación de <i>Linked Data</i>	9
2.3.1. Especificación	10
2.3.2. Modelado	10
2.3.3. Generación	11
2.3.4. Publicación	11
2.3.5. Explotación	11
2.4. Fuentes de Información	12
2.4.1. Bases de Datos	12
2.4.2. Base de Datos Relacional H2	12
2.4.3. Archivos Planos	12
2.4.4. Excel	13
2.4.5. <i>Open Archives Initiative</i> (OAI)	13
2.4.6. <i>Relational database (RDB) to Resource Description Framework (RDF) Mapping Language</i> (r2rml)	13
2.5. Herramientas para el Desarrollo del Framework	14
2.5.1. Pentaho	14

2.5.1.1.	Pentaho Data Integration o Kettle (PDI)	14
2.5.2.	Librería oai2rdf	15
2.5.3.	DB2TRIPLES	15
2.5.4.	Librerías Apache	15
2.5.4.1.	Jena	15
2.5.4.2.	Fuseki	16
2.5.5.	Maven	16
2.5.6.	Elda	16
3.	Análisis y Diseño de los Componentes para el Framework de Generación de RDF	17
3.1.	Introducción	18
3.2.	Selección de los Componentes a Desarrollar para el Framework .	18
3.2.1.	Análisis y Diseño de la Etapa de Especificación	20
3.2.1.1.	Descripción Funcional del Componente de Extracción y Selección de Datos desde Repositorios Digitales OAI	20
3.2.2.	Análisis y Diseño de la Etapa de Modelado	21
3.2.2.1.	Descripción Funcional del Componente de Carga y Selección de Ontologías	21
3.2.3.	Análisis y Diseño de la Etapa de Generación	22
3.2.3.1.	Descripción Funcional del Componente Generación de RDF	22
3.2.4.	Análisis de la Etapa de Publicación	23
3.2.4.1.	Descripción Funcional del Componente de Publicación	24
3.2.5.	Análisis de la Etapa de Explotación	24
3.2.5.1.	Descripción Funcional del Componente para la Explotación de RDF	25
3.3.	Análisis y Diseño de Patrones de Limpieza	25
4.	Implementación de los Componentes para el Framework Enmarcados en la Metodología para la Publicación de <i>Linked Data</i>	27
4.1.	Introducción	28
4.2.	Implementación del Componente para la Extracción de los Datos	28
4.2.1.	Interfaz del Componente <i>OAILoader</i>	28
4.2.2.	Funcionalidad e Implementación de la Interfaz del Componente <i>OAILoader</i>	29
4.2.3.	Implementación de la Salida del Componente <i>OAILoader</i> .	32
4.3.	Implementación del Componente para la Carga de las Ontologías	33
4.3.1.	Interfaz del Componente <i>Get Properties OWL</i>	33
4.3.2.	Funcionalidad e Implementación de la Interfaz del Componente <i>Get Properties OWL</i>	34
4.3.3.	Implementación de la Salida del Componente <i>Get Properties OWL</i>	36

4.4.	Funcionalidad del Componente <i>Ontology Mapping</i>	37
4.4.1.	Interfaz del Componente <i>Ontology Mapping</i>	37
4.4.2.	Salida del Componente <i>Ontology Mapping</i>	38
4.5.	Implementación del Componente para la Generación de Datos en el Estándar RDF	38
4.5.1.	Interfaz del Componente <i>R2RMLtoRDF</i>	38
4.5.2.	Funcionalidad e Implementación de la Interfaz del Componente <i>R2RMLtoRDF</i>	39
4.5.3.	Implementación de la Salida del Componente <i>R2RMLtoRDF</i>	41
4.6.	Implementación del Componente para la Publicación de los Datos	41
4.6.1.	Interfaz del Componente <i>Fuseki Loader</i>	41
4.6.2.	Funcionalidad e Implementación de la Interfaz del Componente <i>Fuseki Loader</i>	43
4.6.3.	Implementación de la Salida del Componente <i>Fuseki Loader</i>	45
4.7.	Implementación del Componente para la Explotación de los Datos	46
4.7.1.	Interfaz del Componente <i>Elda Step</i>	46
4.7.2.	Funcionalidad e Implementación de los Campos de la Interfaz del Componente <i>Elda Step</i>	47
4.7.3.	Implementación de la Salida del Componente <i>Elda Step</i>	49
4.8.	Implementación de los Patrones de Limpieza	50
4.8.1.	Patrón Reutilizable	50
4.8.2.	Patrón no Reutilizable	50
4.8.3.	Código de colores	51
4.8.4.	Notas	52
4.9.	Implementación de la Funcionalidad Utilitaria para los Componentes	52
4.9.1.	Inclusión de Base de Datos H2	52
4.9.1.1.	Tabla OAIPMHDATA	53
4.9.1.2.	Tabla GETPROPERDATA	54
5.	Ejemplo práctico	55
5.1.	Introducción	56
5.2.	Especificación	56
5.2.1.	Configuración del Componente <i>OAILoader</i>	57
5.2.1.1.	Parámetros de Entrada	57
5.2.1.2.	Salida del Componente	58
5.2.2.	<i>Limpieza de Data</i>	59
5.2.2.1.	Bloques de Componentes para la Limpieza en PDI	59
5.3.	Componentes Utilitarios	62
5.3.1.	Configuración Componente <i>Data Precatching</i>	63
5.3.1.1.	Parámetros del Componente	63
5.3.2.	Uso del Componente <i>Block this step until steps finish</i>	63
5.3.2.1.	Parámetros de Entrada	64
5.3.2.2.	Salida del Componente	64

5.4.	Modelado	64
5.4.1.	Configuración del Componente para la Carga de las Onto- logías	64
5.4.1.1.	Parámetros de Entrada	65
5.4.1.2.	Salida del Componente	65
5.4.2.	Configuración del Componente <i>Ontology Mapping</i>	66
5.4.2.1.	Parámetros Base	67
5.4.2.2.	Configuración de la Pestaña <i>Classification</i>	68
5.4.2.3.	Configuración de la Pestaña <i>Anotation</i>	69
5.4.2.4.	Configuración de la Pestaña <i>Relation</i>	71
5.4.2.5.	Salida del Componente	71
5.5.	Generación	72
5.5.1.	Configuración del componente <i>R2RMLtoRDF</i>	72
5.5.1.1.	Configuración De Parámetros	73
5.5.1.2.	Salida del Componente <i>R2RMLtoRDF</i>	74
5.6.	Publicación	74
5.6.1.	Configuración del Componente <i>Fuseki Loader</i>	74
5.6.1.1.	Configuración de Parámetros	75
5.6.1.2.	Salida del Componente <i>Fuseki Loader</i>	76
5.7.	Explotación	77
5.7.1.	Configuración del Componente <i>Elda Step</i>	78
5.7.1.1.	Configuración De Parámetros	78
5.7.1.2.	Salida del Componente <i>Elda Step</i>	79
5.8.	Configuración Finalizada	80
6.	Conclusiones	81
	Anexos	83
7.	Anexos	83
7.1.	Desarrollo de <i>plugins</i> para <i>Pentaho Data Integration</i>	84
7.1.1.	Estructura Basica de los <i>plugins</i> para <i>Pentaho Data Integration</i>	84
7.1.2.	Análisis de las clases de un componente	86
7.1.2.1.	Clase JAVA - OAILoader.java	86
7.1.2.2.	Métodos de la clase OAILoader.java	86
7.1.2.3.	Clase JAVA - OAILoaderData.java	86
7.1.2.4.	Clase JAVA - OAILoaderMeta.java	87
7.1.2.5.	Clase JAVA -Métodos	87
7.1.2.6.	OAILoaderDialog.java	87
7.2.	Configuración del modo de Depuración en <i>Pentaho Data Integration</i>	88
	Bibliografía.	89

Índice de figuras.

2.1.	Resultado de la consulta, “vuelos a Praga mañana por la mañana” [4], en un buscador actual.	5
2.2.	Resultado de la consulta, “vuelos a Praga mañana por la mañana” [4], en un buscador semántico.	6
2.3.	Ejemplo de 3 tripletas en forma de grafo, que representa una tesis que tiene un autor, además el autor tiene un nombre y el nombre es Freddy.	7
2.4.	Ejemplo de 3 tripletas en formato RDF, que representa el grafo anterior de una tesis que tiene un autor, además el autor tiene un nombre y el nombre es Freddy.	7
2.5.	Ejemplo de un grafo RDF representado en formato TURTLE, donde el autor tiene un nombre y el nombre es Fabian.	8
2.6.	FOAF:Core es una ontología utilizada para representar los datos de las personas, aquí se muestra algunos de los metadatos tomados en cuenta por esta ontología.	9
2.7.	Metodología para la publicación de <i>Linked Data</i> [3].	9
3.1.	Figura de <i>Linked Open Data (LOD)</i> [3]	18
3.2.	Diagrama funcional Etapa de Especificación, componente de extracción de datos desde repositorios OAI	21
3.3.	Diagrama funcional Etapa de Modelado. Componente de carga y selección de ontologías [3]	22
3.4.	Diagrama funcional del componente para la Etapa de Generación.	23
3.5.	Diagrama funcional del componente para la Etapa de Publicación. [3]	24
3.6.	Diagrama funcional del componente para la Etapa de Explotación. [3]	25
3.7.	Ejemplo de errores en los datos	26
3.8.	Esquema funcional para la limpieza de los datos	26
4.1.	Interfaz del componente OAILoader	28
4.2.	Estructura valida de una URL y resultado de la invocación a través de un navegador.	29
4.3.	Formatos en un navegador web.	30
4.4.	ComboBox con los formatos.	30

4.5.	Extracto del un registro con el formato xoai.	31
4.6.	XPath obtenidas desde el formato xoai.	32
4.7.	Salida del componente OAILoader	32
4.8.	Interfaz del componente Get Properties OWL	33
4.9.	Boton Add URI al encontrar un namePrefix	34
4.10.	Dialog para seleccionar un ontología desde un archivo local.	35
4.11.	Salida del componente Get Properties OWL	36
4.12.	Interfaz del componente Ontology Mapping	37
4.13.	Interfa del componente R2RMLtoRDF	39
4.14.	FileDialog para elegir el archivo de mapeo.	39
4.15.	Interfaz del componente Fuseki Loader	42
4.16.	Dialogo para seleccionar el RDF desde un archivo local.	43
4.17.	Dialogo para seleccionar un rdf desde un archivo local.	44
4.18.	Servicio levantado en Fuseki Server	45
4.19.	Interfaz del componente Elda Loader	47
4.20.	Ejemplo de configuración de label para la entidad Thesis	48
4.21.	Ejemplo de URL formada para la lista de recursos de la entidad Tesis.	48
4.22.	Transformación aplicado el código de colores.	51
4.23.	Detalle de las correcciones realizadas en cada bloque.	52
4.24.	Esquema DBLOD en la base de datos H2	53
4.25.	Tabla OAIPMHDATA del esquema DBLOD en H2	54
4.26.	Tabla GETPROPERDATA del esquema DBLOD en H2	54
5.1.	Figura Framework para la generación de (LOD) [3]	56
5.2.	Configuración componente OAILoader	57
5.3.	Salida del componente OAILoader	58
5.4.	Transformación ejemplo práctico, componente OAILoader	58
5.5.	Bloque para comprobar la estructura en los datos.	59
5.6.	Configuración del componente Replace in string.	60
5.7.	Ejemplo de una expresión regular evaluando la estructura de los nombres.	61
5.8.	Bloque para evaluar otras estructuras validas en los datos.	61
5.9.	Configuración del componente Modified Java Script Value para corregir duplicados.	62
5.10.	Configuración componente <i>Data Precatching</i>	63
5.11.	Configuración componente Block this step until steps finish	64
5.12.	Configuración componente Get Properties OWL	65
5.13.	Salida del componente Get Properties OWL	66
5.14.	Transformación ejemplo práctico, componente Get Properties OWL	66
5.15.	Configuración componente <i>Ontology Mapping</i>	67
5.16.	Configuración parámetros base del componente Ontology Map- ping	67
5.17.	Configuración Primera Pestaña <i>Clasification</i>	68

5.18. Configuración Segunda Pestaña de Anotación	70
5.19. Configuración Tercera Pestaña	71
5.20. Estado de la transformación ya configurado el componente <i>Ontology Mapping</i>	72
5.21. Configuración del Componente R2RMLtoRDF	73
5.22. Transformación ejemplo práctico, componente R2RMLtoRDF . .	74
5.23. Configuración componente Fuseki Loader	75
5.24. Navegador accediendo al servicio de Fuseki.	76
5.25. Resultado de la consulta SPARQL	77
5.26. Transformación ejemplo práctico, componente Fuseki Loader . .	77
5.27. Configuración Elda Step	78
5.28. Servicio levantado en Elda configurado en este ejemplo para las entidades de tipo Person.	79
5.29. Configuración del Ejemplo práctico Sobre el repositorio ESPOL .	80
7.1. Interfaz gráfica de la plataforma de PDI	84
7.2. Diagrama UML de las clases de un componente de PDI	85
7.3. Proyecto eclipse del componente OAILoader	85
7.4. Declaración del método processRow	86
7.5. Configuración del archivo Spoon.sh	88
7.6. Configuración de Eclipse para la depuración de PDI.	88

desde 1867

Índice de tablas.

3.1. Listado del Análisis Previo a los componentes ya existentes. . . .	19
3.2. Análisis de los Componentes de Salida de PDI.	19



Yo, *Edison Freddy Santacruz Mora*, autor de la tesis *CREACIÓN DE COMPONENTES PARA EL FRAMEWORK DE GENERACIÓN DE RESOURCE DESCRIPTION FRAMEWORK (RDF)*, certifico que todas las ideas, opiniones, y contenidos expuestos en la presente investigación, son de exclusiva responsabilidad de sus autores.


Cuenca, 30 de Octubre 2015.



Edison Freddy Santacruz Mora
C.I. 0401126271

Yo, *Fabian Leonardo Peñaloza Marin*, autor de la tesis *CREACIÓN DE COMPONENTES PARA EL FRAMEWORK DE GENERACIÓN DE RESOURCE DESCRIPTION FRAMEWORK (RDF)*, certifico que todas las ideas, opiniones, y contenidos expuestos en la presente investigación, son de exclusiva responsabilidad de sus autores.


Cuenca, 30 de Octubre 2015.



Fabian Leonardo Peñaloza Marin
C.I. 0104591516

Yo, *Edison Freddy Santacruz Mora*, autor de la tesis *CREACIÓN DE COMPONENTES PARA EL FRAMEWORK DE GENERACIÓN DE RESOURCE DESCRIPTION FRAMEWORK (RDF)*, reconozco y acepto el derecho de la Universidad de Cuenca, en base al Art. 5 literal c) de su Reglamento de Propiedad Intelectual, de publicar este trabajo por cualquier medio conocido o por conocer, al ser este requisito para la obtención de mi título de *Ingeniero de Sistemas*. El uso que la Universidad de Cuenca hiciere de este trabajo, no implicará afección alguna de mis derechos morales o patrimoniales como autor.

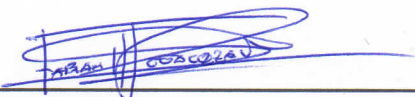
Cuenca, 30 de Octubre 2015.



Edison Freddy Santacruz Mora
C.I. 0401126271

Yo, *Fabian Leonardo Peñaloza Marin*, autor de la tesis *CREACIÓN DE COMPONENTES PARA EL FRAMEWORK DE GENERACIÓN DE RESOURCE DESCRIPTION FRAMEWORK (RDF)*, reconozco y acepto el derecho de la Universidad de Cuenca, en base al Art. 5 literal c) de su Reglamento de Propiedad Intelectual, de publicar este trabajo por cualquier medio conocido o por conocer, al ser este requisito para la obtención de mi título de *Ingeniero de Sistemas*. El uso que la Universidad de Cuenca hiciere de este trabajo, no implicará afección alguna de mis derechos morales o patrimoniales como autor.

Cuenca, 30 de Octubre 2015.



Fabian Leonardo Peñaloza Marin
C.I. 0104591516

Dedicatoria

Dedico este trabajo de tesis primeramente a mi mama Roció y a mis hermanos por su apoyo incondicional a través de los años, que con sus enseñanzas que me han ayudado a no abandonar las cosas aunque sean difíciles. Además agradezco a las personas del Departamento de Ciencias de la Computación que colaboraron en el desarrollo de este proyecto.

Freddy.

Dedicatoria

El presente trabajo de tesis se la dedico de manera especial a mis padres Miguel y María, por su apoyo incondicional en todas las etapas de mi vida, debido que a sus concejos y ejemplo de valor, disciplina y constancia me han motivado siempre a continuar adelante por el camino correcto. A mis hermanos Juan, Johanna y Wilson por la motivación brindada siempre con sus concejos y muestras de afecto. Además agradezco a las personas del Departamento de Ciencias de la Computación que colaboraron en el desarrollo de este proyecto y a nuestro director Ing. Víctor Hugo Saquicela Galarza, PhD.

Fabián.

Agradecimientos

Agradezco de manera especial a mi familia por su apoyo incondicional en cada etapa de mi formación, en especial a mis padres por su ejemplo de esfuerzo diario por obtener sus metas. A todos mis amigos, compañeros y profesores por aportar de gran manera en mi formación con sus enseñanzas las cuales siempre serán duraderas en mi vida .

Fabián.

UNIVERSIDAD DE CUENCA
desde 1867

Agradecimientos

Agradezco de manera especial a mi mama por su apoyo incondicional en cada etapa de mi vida. A todos mis amigos por compartir momentos buenos y malos en nuestra etapa universitaria, también agradezco a mis profesores por su aporte en mi crecimiento personal y académico.

Freddy.

UNIVERSIDAD DE CUENCA
desde 1867

Capítulo 1

Introducción

La web semántica surge por la necesidad de definir semánticamente la gran cantidad de información disponible en internet tanto para máquinas como para humanos. Para cumplir este objetivo se utiliza el estándar **Resource Description Framework** RDF, el cual es un mecanismo flexible para representar información usando recursos. Sin embargo, la generación de RDF actualmente tiene una gran complejidad debido a que existen varias metodologías y se requiere altos conocimientos de tecnologías semánticas.

Actualmente el crecimiento de la web semántica alrededor del mundo ha generado que investigadores busquen la manera de agilizar el proceso de *Linked Open Data* (LOD), entre sus aportes tenemos: *D2QR* que es una plataforma que sigue LOD, a través de diferentes aplicaciones para cumplir cada una de las etapas. *D2QR* en la etapa de especificación utiliza una base de datos relacional para obtener los datos, en la etapa de modelado utiliza *D2RQ Mapping Language* para relacionar las ontologías con los datos, en la etapa de generación utiliza *dump-rdf* para generar el RDF, en la etapa de publicación utiliza *D2QR Server* que genera un SPARQL Endpoint y finalmente en la etapa de explotación utiliza *D2RQ Platform* la cual explota el SPARQL Endpoint. También se tiene *Open Refine* que es una herramienta que trabaja con datos desordenados mediante tres etapas, las cuales son: exploración de datos, limpieza y transformación de datos, además analiza coincidencias en los datos. Otro enfoque propuesto es: *Non-Ontological Resources to Ontologies* (NOR2O) que transforma datos no ontológicos en ontologías aplicando ingeniería inversa, esta tecnología no sigue LOD. Aunque los enfoques solucionan de manera parcial la generación de LOD, estos no proveen una interfaz amigable al usuario, por esta razón requieren un alto conocimiento para ser utilizadas; Por lo tanto, un enfoque más adecuado para automatizar este

proceso se presenta en este proyecto, el cual tiene como objetivo proveer herramientas de fácil uso en cada etapa de la metodología LOD.

La no existencia de una herramienta que permita la conversión fácil y rápida de datos al estándar RDF siguiendo la metodología LOD, ha motivado el desarrollo de un Framework que siga el proceso de LOD y permita la extracción de los datos de diferentes fuentes, carga de las ontologías con sus respectivos vocabularios, el mapeado entre los datos y ontologías, la generación de datos en RDF, la publicación de los datos y la explotación de los mismos cumpliendo esta metodología. Ahora bien, este Framework es desarrollado bajo la tecnología JAVA, utilizando como plataforma Pentaho Data Integration (PDI), la cual permite integrar todo el proceso de LOD, además proporciona una interfaz gráfica intuitiva y de fácil uso en cada etapa, logrando abstraer la complejidad que conlleva este proceso y optimizando el tiempo de los desarrolladores, para lograr nuevos avances acerca de esta tecnología, y así dar solución a esta problemática.

Capítulo 2

Marco Teórico



UNIVERSIDAD DE CUENCA
desde 1867

2.1. Introducción

En este capítulo se aborda los fundamentos teóricos necesarios para el desarrollo de los componentes dentro del Framework de generación de RDF, incluyendo conceptos básicos sobre las metodologías y herramientas.

2.2. Web Semántica

En este apartado se tratarán los temas concernientes a la web semántica.

2.2.1. ¿Qué es la Web Semántica?

La Web Semántica, es una extensión de la Web actual [20], su principal objetivo es proporcionar mayor significado a los datos, permitiendo a los usuarios búsquedas con resultados más exactos. Esto se consigue con la Web Semántica dado que los datos están estandarizados y unificados, lo cual facilita el proceso a los buscadores [4]. Actualmente los motores de búsquedas devuelven miles de resultados al realizar una consulta acerca de un tema específico, debido a que las peticiones se realizan sobre datos sin ningún estándar, lo que no facilita la ejecución de las consultas.

2.2.2. ¿Para qué Sirve la Web Semántica?

El uso de la Web se ha vuelto parte de la vida cotidiana, cambiando la manera que se realiza diferentes actividades tales como: negocios, pagos de servicios básicos, transacciones bancarias, las comunicaciones, entretenimiento, etc... Aunque para facilidad de los usuarios se han creado diferentes aplicaciones dentro de la web para atender estas actividades, lo cual permite a los usuarios tener acceso, crear nueva información de manera acelerada y de manera heterogénea. Información que con el paso del tiempo, al tener grandes cantidades y con heterogeneidad tanto sintáctico como semántico, han derivado en los problemas de la Web actual. La Web Semántica se utiliza para dar solución a estos problemas mediante diferentes tecnologías que permiten estandarizar, unificar y dar significado a los datos disponibles en la Web actual. [4] En conclusión la Web Semántica sirve para el manejo de grandes cantidades de datos y la homogeneización los mismos.

2.2.3. ¿Cómo Funciona la Web Semántica?

La Web Semántica funciona con datos estandarizados y enriquecidos con semántica, con lo cual responde a las consultas de manera exacta. Actualmente los buscadores retornan cualquier información relacionada al tema consultado. Por ejemplo, si buscamos “Los vuelos a Praga para mañana por la mañana” [4], esta consulta devolverá información relacionada con alguna palabra, en este caso Praga, pero no exactamente lo solicitado. Se muestra en la figura [2.1] el resultado de la consulta en un buscador actual. Los datos enriquecidos con semántica permitirán a los buscadores semánticos devolver resultados exactos, porque cada palabra en “la oración como mañana adquirirían significado, convirtiéndose en un día concreto calculado en función de un hoy. Algo semejante ocurriría con el segundo mañana, que sería interpretado como un momento determinado del día. Todo ello a través de una Web en la que los datos pasan a ser información llena de significado. El resultado final sería la obtención de forma rápida y sencilla de todos los vuelos a Praga para mañana por la mañana.” [4]. Se muestra en la figura [2.2] el resultado de la consulta en un buscador semántico.

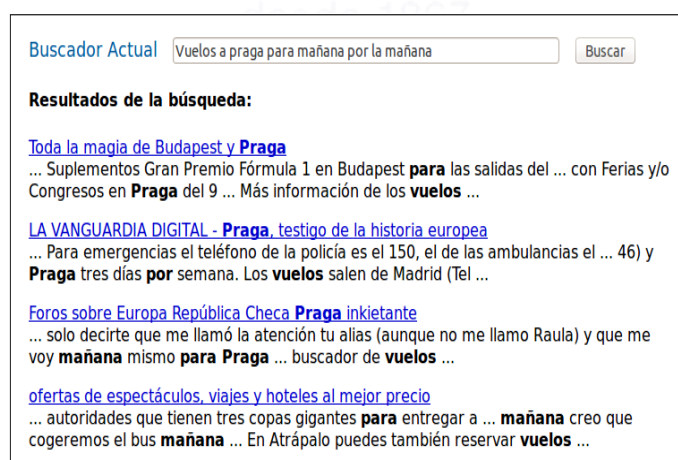


Figura 2.1: Resultado de la consulta, “vuelos a Praga mañana por la mañana” [4], en un buscador actual.

La forma en la que se procesará esta información no sólo será en términos de entrada y salida de parámetros sino en términos de su SEMÁNTICA. La Web Semántica como infraestructura basada en metadatos aporta un camino para razonar en la Web, extendiendo así sus capacidades.[4]



Figura 2.2: Resultado de la consulta, “vuelos a Praga mañana por la mañana” [4], en un buscador semántico.

La obtención de resultados en los buscadores semánticos no es mágico, los computadores lo realizan porque tienen operaciones definidas que actuarán sobre datos bien definidos. Para conseguir datos correctamente definidos, la Web Semántica utiliza principalmente tres tecnologías: la primera denominada *Resource Description Framework* (RDF), la segunda *Query Language for RDF* (SPARQL) y la tercera *Web Ontology Language* (OWL), estas herramientas colaboran en la creación de una infraestructura donde será posible compartir y reutilizar los datos.

2.2.4. Herramientas de Web Semántica

En esta sección se detalla las principales herramientas utilizadas en la Web Semántica para la descripción, estandarización y consulta de los datos.

2.2.4.1. *Resource Description Framework* (RDF)

RDF es un estándar para representar metadatos a través de un marco común de trabajo, que permiten el intercambio de los datos en la Web semántica, brindando facilidades para relacionarse entre datos de diferentes fuentes. Además es muy adaptable y un modelo definido puede evolucionar sin necesidad de cambiar los datos [18]. RDF para identificar los recursos usa los *Uniform Resource Identifiers* (URIs), que describen los recursos en términos de propiedades simples y valores, que se conocen como tripletas. Una tripleta es una estructura que tiene un Sujeto, un Predicado y Objeto con lo cual se forma un grafo, el cual se puede

procesar y manipular utilizando diferentes herramientas. Se muestra en la figura [2.3] un ejemplo de 3 grafos que representa una Tesis que tiene un Autor, también este Autor tiene un Nombre y el Nombre es Freddy aquí se ha representado 3 tripletas para representar recursos.

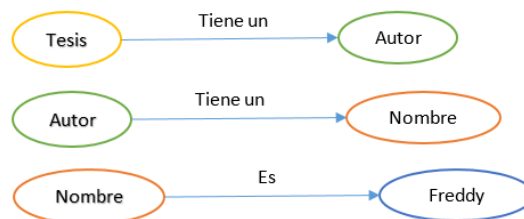


Figura 2.3: Ejemplo de 3 tripletas en forma de grafo, que representa una tesis que tiene un autor, además el autor tiene un nombre y el nombre es Freddy.

```

<http://example.org/#Tesis>
<http://www.bibliotecasExample.com/schemas/relationship/Tieneun>
<http://example.org/#Autor>

<http://example.org/#Autor>
<http://www.bibliotecasExample.com/schemas/relationship/Tieneun>
<http://example.org/#Nombre>

<http://example.org/#Nombre>
<http://www.bibliotecasExample.com/schemas/relationship/Es>
<http://example.org/Freddy>
  
```

Figura 2.4: Ejemplo de 3 tripletas en formato RDF, que representa el grafo anterior de una tesis que tiene un autor, además el autor tiene un nombre y el nombre es Freddy.

2.2.4.2. *Terse RDF Triple Language* (Turtle)

Es una definición de sintaxis concreta para RDF, la cual permite escribir completamente un Grafo RDF, de una manera compacta y textual. Los archivos en este formato se reconocen por tener la extensión TTL [2].

```
@base <http://example.org/> .  
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
@prefix rel: <http://www.perceive.net/schemas/relationship/> .  
  
base:Tesis rel:Tieneun base:Autor .  
  
base:Tesis rel:Tieneun base:Nombre.  
  
base:Nombre rel:Tieneun base:Fabian.  
  
# los comentarios se hacen con numeral
```

Figura 2.5: Ejemplo de un grafo RDF representado en formato TURTLE, donde el autor tiene un nombre y el nombre es Fabian.

2.2.4.3. *Query Language for RDF (SPARQL)*

SPARQL es un lenguaje estandarizado por la W3C para bases de datos RDF, el cual define la sintaxis, semántica y léxico para realizar consultas, inserciones y eliminación de datos. SPARQL trabaja sobre un esquema RDF cliente servidor (SPARQL Endpoint), en el cual el cliente envía una operación de consulta al servidor, este la procesa y retorna un resultado. [14]

Entre las principales características de SPARQL están:

- Diferentes formatos para los datos de salida.
- Actualización de grafos RDF.
- Protocolo para RDF, usado en él envió de peticiones mediante HTTP o SOAP
- Descubrir información acerca de la datos almacenados (Dataset).
- Lenguaje de consulta.

2.2.4.4. *Web Ontology Language (OWL)*

Una ontología define los términos para describir y representar un área del conocimiento. Las ontologías son usadas por personas, bases de datos y aplicaciones que necesitan compartir un dominio común de información, por ejemplo medicina, matemáticas, datos personales, etc... Los metadatos utilizados para estructurar una área de conocimiento que están enfocados en convertirse en un dominio que pueda ser reutilizado por los demás usuarios [10]. Se muestra en la figura [2.6] un ejemplo de ontología usada para representar información de las personas.

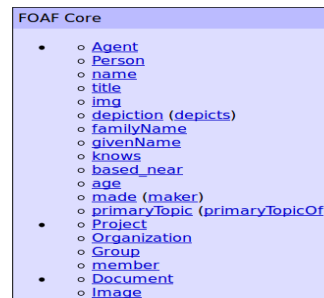


Figura 2.6: FOAF:Core es una ontología utilizada para representar los datos de las personas, aquí se muestra algunos de los metadatos tomados en cuenta por esta ontología.

2.3. Metodología para la Publicación de *Linked Data*

La metodología *Linked Open Data* (LOD) para publicación de *Linked Data* consiste de un conjunto de pasos con decisiones tecnológicas y de diseño. Se definen como una serie de directrices metodológicas involucradas con este proceso que son el resultado de la experiencia en la producción de *Linked Data* en varios contextos Gubernamentales.[3] Ver figura 2.7.

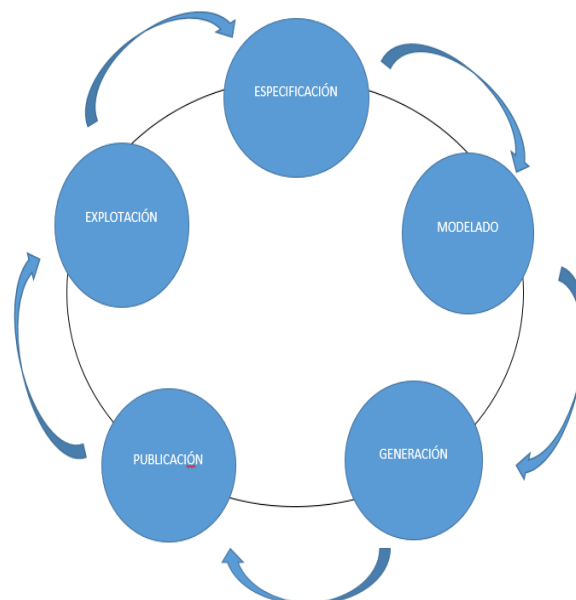


Figura 2.7: Metodología para la publicación de *Linked Data* [3].

2.3.1. Especificación

Esta fase obtiene una especificación detallada de los requisitos dando como resultado una memoria proporciona varios beneficios tales como:

- (a) Establecer actividades entre proveedores y clientes.
- (b) Obtener una base para estimar costos y tiempos.
- (c) Reducción de esfuerzos en el desarrollo [3].

Los pasos en esta actividad son:

- Identificación y Análisis de Fuentes de Datos: Se definen claramente las fuentes de los datos y sus formas de alojamiento.
- Diseño de URIs: En *Linked Data* se utiliza las URI's para la identificación de elementos y enlaces a otras fuentes, por lo tanto se deben elegir las más adecuadas para cada caso particular.
- Definición de la Licencia: Existen 5 niveles de licencia que se deben elegir para la publicación, en donde entre menos grado de compromiso legal será más fácil automatizar. Niveles: [3]
 - (a) Licencia UK Open Government.
 - (b) Licencia Open Database.
 - (c) Licencia Public Domain Dedication.
 - (d) Licencia Open Data Commons Attribution.
 - (e) Licencia Creative Commons.

2.3.2. Modelado

En esta actividad se determina la ontología que se utilizará para modelar las fuentes de datos. Se recomienda utilizar vocabularios y ontologías ya existentes solo en el caso de no existir una ontología adecuada se debe modificar o crear una nueva según las necesidades requeridas.

En la actividad de selección de una ontología se recomienda los siguientes repositorios para encontrar vocabularios adecuados: Schema Web, Schema Cache y Swoogle. En el caso de ser necesaria la creación de una nueva ontología existen herramientas como Protege o Neologism.[3]

2.3.3. Generación

Esta fase consta de 3 tareas: transformación, limpieza de datos y enlace.

- Transformación: Se considerada en esta parte la transformación de todo el contenido de origen de datos en una instancia RDF.
- Limpieza de Datos: Consiste en dos etapas.
 1. Identificar los posibles errores comunes como por ejemplo usar espacios de nombres (namespaces) sin vocabularios.
 2. Corrección de todos los errores identificados anteriormente.
- Enlace: Incluir enlaces a otras URIs donde se pueda extender y compartir información con otras fuentes de datos. Creación de enlaces entre el conjunto de datos interno y conjuntos de datos externos.[3]

2.3.4. Publicación

Esta fase consiste en 3 actividades.

- Publicación de datos: una vez generado el RDF se almacena en un triplestore como por ejemplo: Virtuoso, Jena SDB o Sesame Native.
- Publicación de metadatos: se incluye información de los datos mediante vocabularios tales como VoID entre otros.
- Habilitar la detección eficaz: esta actividad define una sincronización efectiva del conjunto de datos.[3]

2.3.5. Explotación

En esta actividad se tiene como premisa la transparencia para ofrecer aplicaciones y fomentar el uso y reuso público o comercial de la información.

Además esta fase permite el desarrollo de aplicaciones de alto nivel que realicen la explotación de los datos y provea una rica interfaz para cada usuario.[3]

2.4. Fuentes de Información

Una fuente de información es cualquier medio que permita almacenar y recuperar los datos, tales como: bases de datos, archivos de texto plano, repositorios digitales, etc... Estos contienen grandes cantidades de información heterogénea de temas. Por lo cual, se detallará cada fuente en la siguiente sección.

2.4.1. Bases de Datos

Una base de datos permite almacenar, modificar, eliminar y recuperar datos que son almacenados en memoria secundaria, los cuales la base de datos los organiza a través de diferentes estructuras de lógicas. Además una base de datos tiene diferentes enfoques para lograr almacenar la información dependiendo de las necesidades que tenga una organización, por esta razón existen: las bases relacionales, documentales, geográficas etc... Dependiendo del propósito que persigan las empresas. En conclusión una base de datos se ajusta a las necesidades de la organización dando solución a sus requerimientos, ya sea con soluciones de bases de datos de pago o gratuitas. [11].

Los SGBD más conocidos son:

- MySql
- PostgreSQL
- Oracle

2.4.2. Base de Datos Relacional H2

Es un gestor de base de datos relacional desarrollado en Java. Una de las características más importantes es que se puede integrar en aplicaciones Java para ser accedido directamente con SQL sin necesitar una conexión a través de sockets. Finalmente se debe recalcar que esta disponible con software libre bajo la licencia publica de Mozilla. [12]

2.4.3. Archivos Planos

Un archivo plano es una representación digital de la mayoría de documentos físicos como son: cartas, tesis de grado, oficios, libros, revistas, etc... estos

archivos únicamente almacenan información escrita no imágenes, la cual es entendida por las maquinas si estos son guardados en formatos binarios que solo estas pueden leer o por el hombre si la información es guardada en texto plano. Los archivos son la forma más básica de almacenamiento de datos, porque inicialmente eran las bases de datos de los primeros programas informáticos ya que guardaban la información en formato de bits que era leída por estos únicamente. Un archivo plano está compuesto de dos partes ejemplo: ArchivoPrueba.txt, la primera parte representa el nombre archivo, el cual facilita su identificación y la segunda parte es la extensión que ayuda a identificar en algunos sistemas operativos (SO), que software puede manipular este tipo archivo. [19]

2.4.4. Excel

Excel es un programa del paquete de ofimática de Microsoft que permite realizar diferentes tipos de cálculos matemáticos, gráficos estadísticos entre otras funciones de manera fácil e intuitiva. Además es utilizado para almacenar grandes cantidades de información como si fuera una base de datos. [22]

2.4.5. *Open Archives Initiative (OAI)*

OAI es una iniciativa para la creación de repositorios de archivos abiertos de documentos educativos, estos permiten la interoperabilidad e intercambio de datos, a través del uso de metadatos (denominados metadatos harvesting), para su edición y almacenamiento entre los repositorios.[7]. OAI trabaja con el protocolo OAI-PMH el cual provee el mecanismo para la extracción de registros que contienen metadatos desde los repositorios OAI, este mecanismo se llama Data Provider el cual utiliza los estándares HTTP (*Hypertext Transport Protocol*) y XML (*Extensible Markup Language*) para responder las peticiones. Las respuestas que están en formatos establecidos por la comunidad desarrolladora, por ejemplo: rdf, xoai, dim, qdc etc. [13]

2.4.6. *Relational database (RDB) to Resource Description Framework (RDF) Mapping Language (r2rml)*

Es una recomendación de la *World Wide Web Consortium* (W3C) que permite la conversión de los datos almacenados en bases de datos relacionales a RDF

dataset ¹. Esto se realiza con la utilización del lenguaje de mapeo que provee r2rml, que permite relacionar los datos de una base relacional con el vocabulario de ontológico que el autor ha escogido. En conclusión la entrada que necesita es una base de datos relacional, que al utilizar el archivo de mapeo r2rml da como resultado un archivo con un Dataset en formato RDF [6].

2.5. Herramientas para el Desarrollo del Framework

En esta sección se detalla las herramientas utilizadas en el desarrollo del Framework.

2.5.1. Pentaho

Pentaho es una suite de productos de Business Intelligence (BI) que proveen integración de datos, servicios de análisis de datos, reportes, cuadros de mando, minería de datos y ETL (extracción, transformación y carga de datos), que se enfoca en proporcionar herramientas de fácil manipulación con interfaz gráfica. Son utilizadas por un gran número de usuarios que realizan BI, brindándoles facilidad en la visualización de los resultados de sus negocios. Todos estos productos son de software libre y desarrollados con el lenguaje de programación JAVA. [15]

2.5.1.1. Pentaho Data Integration o Kettle (PDI)

PDI es el componente de la suite Pentaho encargado de los procesos de extracción, transformación y carga (ETL) [5], La mayoría de ocasiones son utilizadas en el desarrollo de DataWarehouse, pero PDI también puede ser usado para otras actividades como:

- Migración de datos entre aplicaciones o bases de datos.
- Exportación de datos desde bases de datos a archivos planos.
- Carga de grandes cantidades de datos en bases de datos.
- Limpieza de datos.
- Integración de aplicaciones.

¹RDF dataset: expresa la información como tripletas con sujeto, predicado y objeto

PDI es la plataforma que aloja los plugins desarrollados en este proyecto, ya que está facilita la integración de nuevos componentes con otras funcionalidades.

2.5.2. Librería oai2rdf

Es una herramienta de software que permite extraer datos desde repositorios digitales OAI, para transfórmalos al estándar RDF. [17]

2.5.3. DB2TRIPLES

Es una librería de software que sirve para la extracción de información de bases de datos relacionales, que posteriormente se almacenan en RDF triplestore. Este proceso lo realiza utilizando un estándar de mapeo llamado r2rml, el cual permite integrar los modelos ontológicos y los datos de forma directa. Además, una ventaja de db2triples es tener una licencia LGPL.[1]

2.5.4. Librerías Apache

A continuación se describen las librerías ocupadas en este proyecto pertenecientes a Apache.

2.5.4.1. Jena

Es una librería abierta y gratuita de JAVA que sirve como un marco trabajo para la construcción de aplicaciones basadas en ontologías. Se caracteriza por su arquitectura que incluye: [8]

- Una API para leer, procesar y escribir ontologías RDF y OWL.
- Un motor de inferencia para razonar sobre fuentes de datos RDF y OWL.
- Estrategias de almacenamiento flexible para almacenar un gran numero tripletas RDF en memoria o fichero eficientemente.
- Motor de consultas ARQ compatible con las especificaciones SPARQL.
- Servidores para alojar datos RDF a ser publicados en otras aplicaciones usando una variedad de protocolos.

2.5.4.2. Fuseki

Fuseki es un servidor SPARQL desarrollado por apache JENA. Se accede mediante servicios REST sobre HTTP, su característica principal es brindar las interfaces correspondientes para los siguientes estándares:

- SPARQL Query
- SPARQL Update
- SPARQL Protocol
- SPARQL Graph Store HTTP Protocol

Fuseki permite correr estos servicios sobre la máquina local en la que se ejecute, donde se debe tener cargado y configurado el Dataset a trabajar.[9]

2.5.5. Maven

Es una herramienta de software para la gestión y construcción de proyectos JAVA. Tiene un modelo de construcción parecido a Apache ANT pero con un modelo de configuración más simple basado en XML que permite una construcción eficaz.

Maven utiliza como base un Project Model Object (POM) que describe el orden de construcción del proyecto de software, la dependencia de otros módulos o componentes externos en la Web. Una característica principal de Maven es poseer un motor para funcionar en la web el cual permite descargar dinámicamente plugins de un repositorio que provee muchas versiones de diferentes proyectos Open Source de Java, Apache u otras organizaciones de desarrollo. [21]

2.5.6. Elda

Elda es una implementación JAVA de una API *Linked Data* que permite al usuario tener una interfaz configurable de estilo REST para acceder a un contenedor de tripletas RDF (triplestore).

Es usada ampliamente por desarrolladores que mediante tecnologías web como JavaScript y Json pueden acceder a su datos RDF y desplegarlos en un navegador web. Una ventaja a considerar es que Elda esta licenciado bajo la Open Source de Apache permitiendo un uso libre y abierto. [16]

Capítulo 3

Análisis y Diseño de los Componentes para el Framework de Generación de RDF



3.1. Introducción

En este capítulo se tratará el análisis para el desarrollo de un Framework para la generación de *Linked Data*, donde se tomará como guía cada una de las etapas descritas en la metodología **Linked Open Data (LOD)**. Entonces, el objetivo del capítulo es analizar cada etapa de la metodología y diseñar un componente acorde a sus requerimientos, empezando desde la especificación de sus funcionalidades individuales hasta la manera de integrarse con los otros componentes.

3.2. Selección de los Componentes a Desarrollar para el Framework

La figura *Linked Open Data (LOD)* es el esquema para el desarrollo del Framework sobre Pentaho Data Integration, en donde para cada etapa se muestra los componentes requeridos y como interactuarán entre si para su funcionamiento colectivo. Finalmente la definición que se realizara para cada componente estará sustentada en este esquema.

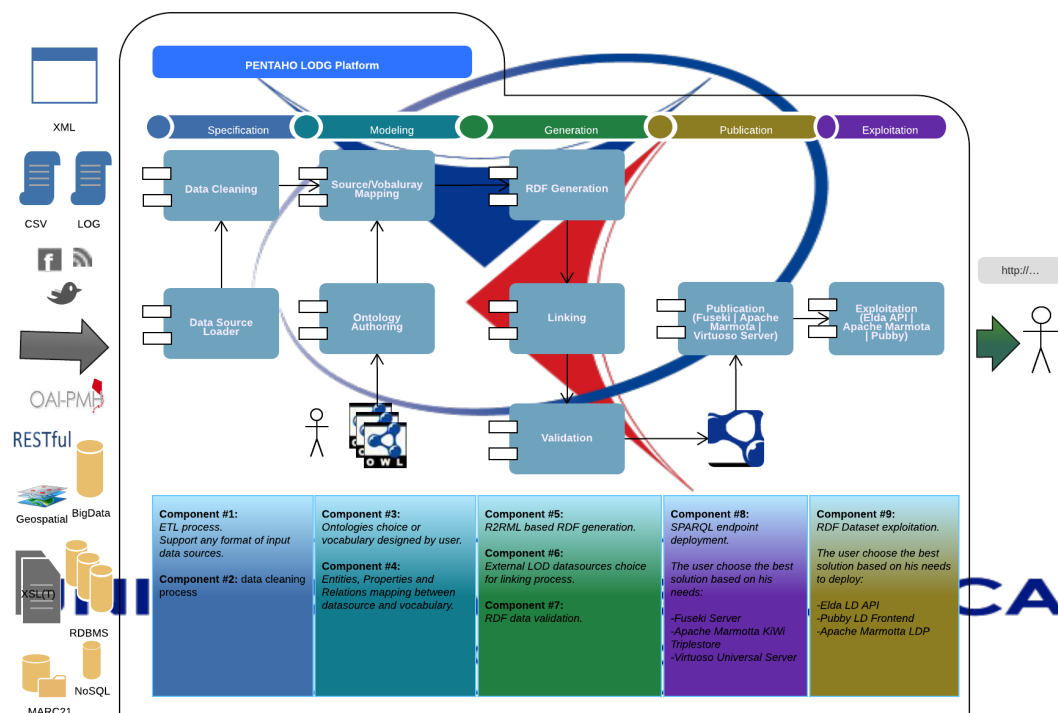


Figura 3.1: Figura de *Linked Open Data (LOD)* [3]

En este apartado siguiendo la metodología LOD se realiza un breve análisis de las necesidades de cada etapa: en la primera etapa se requiere una herramienta que permita la extracción y selección de los datos desde servicios OAIPMH; en la etapa de modelado se requiere un componente que permita la carga y selección de las ontologías; para la etapa de generación se requiere una herramienta que convierta los datos al formato RDF; en las etapas de publicación y explotación se requiere componentes que configuren automáticamente herramientas para publicar y explotar el RDF. Por lo tanto se analiza la herramienta PDI, para establecer si los componentes existentes solventan las necesidades de cada etapa, como muestran las tablas [3.1] y [3.2].

Nombre Plugin	Descripción	Salida
RSS Output	Escribir la entrada proporcionada en el estándar RSS	archivo RSS
Rss Output	Trasforma los datos en archivos JSON	archivo JSON
ArffOutput	Almacena los datos en un archivo ARFF de WEKA	archivos WEKA
Excel Output	Exporta los datos a un archivo de Excel	archivos Excel
Table Output	Inserta los datos en una tabla BD	Tabla de base da datos

Tabla 3.1: Listado del Análisis Previo a los componentes ya existentes.

Nombre Componente	Descripción	Entrada
CSV file input	Lee los datos desde una archivo separado por comas	Archivo.csvs
Microsoft Excel Input	Lee los datos desde una archivo Microsoft Excel	Archivos.xlsl
Microsoft Access Input	Lee los datos desde una base de datos access	Archivo “MDB”
RSS Input	Obtiene los datos desde un url de noticias	url RSS feeds
Table input	Lee los datos desde cualquier base de datos	Conexión a bases de datos
Text file input	Lee los datos desde un archivo plano	Archivo plano
Get data from XML	Lee los datos desde un archivo con formato XML	Archivo.xml
Json Input	Lee los datos desde un archivo con formato JSON	Archivo.json
Web Services Lookup	Extrae los datos desde un servicio web	url WSDL

Tabla 3.2: Análisis de los Componentes de Salida de PDI.

Después de realizar un análisis exhaustivo de los componentes de entrada y salida de PDI, se ha determinado que no existen componentes que solventen las necesidades que se requieren en cada etapa del proceso de generación de RDF, por lo tanto es necesario el desarrollo de nuevos componentes para PDI que ayuden a cumplir los objetivos de la metodología LOD en sus etapas.

3.2.1. Análisis y Diseño de la Etapa de Especificación

De acuerdo a la metodología para la publicación de *Linked Data*, en esta etapa se identifica y analiza las fuentes de datos, que para efecto de este proyecto son repositorios digitales OAI, los cuales almacenan información académica generada a través de los años en las instituciones educativas tales como: tesis, trabajos finales de graduación, libros, artículos, resultados de investigación, trabajos docentes, etc.. Adicionalmente, el análisis previo a los componentes de entrada de PDI que permiten la carga de datos de las diferentes fuentes, ha determinado la necesidad del desarrollo de un nuevo componente para PDI que permita la extracción de los datos desde este tipo de servidor.

3.2.1.1. Descripción Funcional del Componente de Extracción y Selección de Datos desde Repositorios Digitales OAI

La información almacenada en repositorios digitales OAI está organizada en diferentes formatos que soportan este tipo de repositorios, como son: xoai, oai_dc, uketd_dc, dim, etdms, marc, qdc, rdf, ore, mods, mets, didl, los cuales son estructurados con XML. Todos estos formatos son metadatos OAI-PMH que permiten la visualización, consulta y distribución a nivel mundial de los datos, tales como: tesis, trabajos finales de graduación, libros, artículos, resultados de investigación, trabajos docentes, etc... Es decir cualquier documento académico de una institución educativa. Los metadatos OAI-PMH son estándares que utilizan los encargados de generar nuevos repositorios con el objetivo de tener una estructura común que sea compartida entre las demás instituciones.

Como se muestra en la figura [3.1], el componente 1. *Data Source Loader*, recibe los datos desde un fuente, en este caso es una URL de un repositorio OAI, que proporciona acceso a los datos contenidos en este. Los datos extraídos son devueltos como XML, con una estructura diferente de acuerdo a los formatos que contenga el repositorio. Por lo tanto, el componente debe permitir seleccionar el formato del cual se obtendrá los datos. Adicional, la extracción de los datos desde el XML, se realizan de manera general donde se muestren todos los campos o parcial donde se especifique que campos se quiere obtener desde los registros del repositorio. En consecuencia el componente debe dar la facilidad de especificar como se realiza la extracción de los campos. Finalmente el componente debe dar como resultado una tabla con tres columnas, la primera corresponde al identifi-

cador que contienen los registros, la segunda columna contiene la descripción de cada campo de los registros y la tercera columna contiene la información de cada registro.

El esquema funcional general del componente se muestra en la figura 3.2 donde se indica el proceso general que realiza el componente.

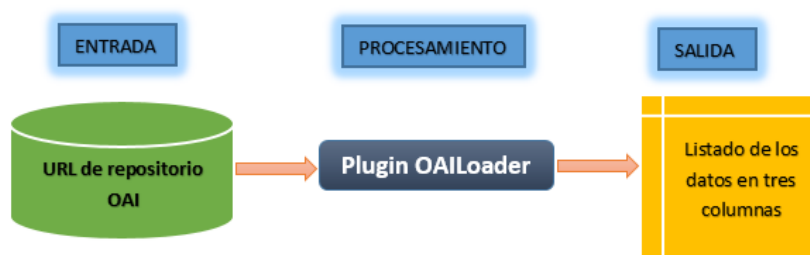


Figura 3.2: Diagrama funcional Etapa de Especificación, componente de extracción de datos desde repositorios OAI

3.2.2. Análisis y Diseño de la Etapa de Modelado

La etapa de modelado es la actividad que determina las ontologías y vocabularios adecuados a utilizar, para relacionarse con los datos obtenidos en la etapa de especificación. Para conocer si una ontología es adecuada se debe tener en cuenta las clases, propiedades que esta contiene y determinar si estas permite especificar cada uno de los datos del dominio que se pretende modelar. Por lo tanto, se requiere un componente que permita seleccionar ontologías y listar sus clases y propiedades. De acuerdo al análisis previo de PDI descrito en tabla 3.1, no existe un componente que enmarque los requerimientos que esta etapa de la metodología necesita, esto justifica el desarrollar un componente que permita la carga y selección de ontologías adecuadas para el dominio de los datos. [3]

3.2.2.1. Descripción Funcional del Componente de Carga y Selección de Ontologías

Como se muestra en la figura 3.1, el componente recibirá como entrada un archivo local o la URL de una ontología en la web para la cual se desea cargar su estructura es decir, se listara todas las propiedades y clases con el fin de que el usuario pueda realizar una selección, dejando únicamente las que considere oportunas para el dominio de los datos. Por último, el componente deberá entregar

una tabla con información de todas las ontologías que fueron seleccionadas con sus correspondientes propiedades y clases.

Un esquema general del componente se muestra en la figura 3.3 donde se indica el proceso general que realiza el componente.



Figura 3.3: Diagrama funcional Etapa de Modelado. Componente de carga y selección de ontologías [3]

3.2.3. Análisis y Diseño de la Etapa de Generación

En esta etapa de la metodología de publicación de *Linked Data*, se toma los datos extraídos en la etapa de especificación, las clases y propiedades de las ontologías seleccionadas en la etapa de modelado, que a través del proceso de *Mapping* en la etapa de modelado permite relacionar los datos con las ontologías. En la figura 3.1 se muestra el componente 3, *Source to Ontology Mapping*¹. Esto sirve de punto de partida para la generación de tripletas del estándar RDF. Por otra parte, el análisis previo de los componentes de salida de PDI que ilustra la tabla 3.1, determina que no existe un componente que realice la tarea de generar RDF dentro de PDI, por lo tanto, es necesario el desarrollo de un componente que solvete las necesidades de esta etapa.

3.2.3.1. Descripción Funcional del Componente Generación de RDF

El mapeo entre los datos y las ontologías es importante en el siguiente paso en la metodología de publicación de *Linked Data*, en la cual hay que generar los datos en RDF. Esta generación se enfoca en la creación de un archivo que represente el formato RDF, el cual organiza los datos en tripletas que contengan un sujeto, un predicado y un objeto que describen un recurso. Por lo tanto, este componente crea las tripletas que correspondan a cada uno de los atributos en el dominio de los datos que se está modelando.

¹Componente 3: este componente mapea las clases y propiedades de las ontologías con cada uno de los atributos de los datos.

El componente de generación en la figura 3.1, se encuentra en la sección *Generation* como el componente 4: *RDF Generation*, este utiliza la salida del componente 3, el cual da como salida un archivo con los datos mapeados con las ontologías ². Por lo tanto, el componente de generación toma como entrada el archivo de mapeo y genera como resultado un archivo con los datos en formato RDF. Además el componente permite escoger el directorio donde será la salida del archivo de generación.

En la figura 3.4 se muestra el esquema funcional que debe cumplir el componente para la generación de RDF, para alcanzar el objetivo de esta etapa.



Figura 3.4: Diagrama funcional del componente para la Etapa de Generación.

3.2.4. Análisis de la Etapa de Publicación

Esta actividad de la metodología LOD permite publicar la información RDF obtenida de la etapa de generación, para lo cual se debe cargar un conjunto de datos RDF en un triplestore que sea accesible a los usuarios desde un SPARQL Endpoint. Por lo que es necesario contar con un componente donde se reciba como entrada un RDF, y como salida despliegue un servicio que permita acceder a un SPARQL Endpoint creado en un triplestore local. Además el servidor que solventa este requisito debe ser compatible con JAVA y de arquitectura abierta para poder ser incorporado al componente en cuestión. Después de un análisis de los requerimientos se llegó a la conclusión de que el servidor Fuseki de Apache cumple a cabalidad los requerimientos planteados y por ende resulta idóneo para el desarrollo de este componente.

Se realizó un análisis en busca de alguna herramienta ya existente en PDI que pueda realizar la publicación RDF. Pero como se muestra la tabla 3.1 no existe algún componente que brinde alguna de las funcionalidades buscadas.

²Nota: el componente 3 no es desarrollado como parte de este proyecto de tesis.

3.2.4.1. Descripción Funcional del Componente de Publicación

Como se muestra en la figura 3.1 el componente recibe como entrada un RDF creado en la etapa de generación, lo siguiente es permitir la configuración de Fuseki mediante una interfaz amigable, dejando únicamente las opciones apropiadas para el mismo. Una vez validada la configuración el usuario podrá elegir un directorio para generar la instancia del Fuseki con el triplestore creado. Por último, el componente permitirá desplegar un servicio que ejecuta el servidor Fuseki creado localmente en un navegador web. Un esquema funcional del componente se muestra a continuación en la figura 3.5 donde se representa el proceso antes mencionado.

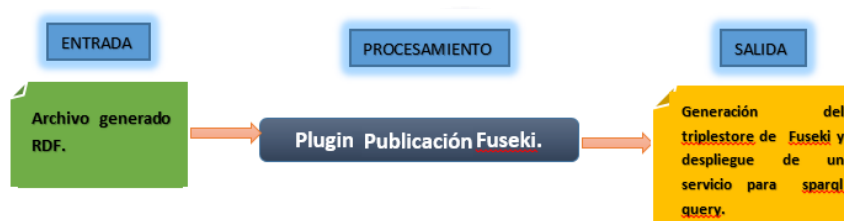


Figura 3.5: Diagrama funcional del componente para la Etapa de Publicación. [3]

3.2.5. Análisis de la Etapa de Explotación

Para la fase de explotación en la metodología de *Linked Data* se busca fomentar el uso de la información. Se requiere un componente que ofrezca transparencia al usuario para permitir el acceso a un SPARQL Endpoint que consuma al triplestore generado en la etapa de publicación.

Existen varias herramientas para realizar la publicación de un triplestore pero se ha elegido Elda De Epimorphics por las siguientes razones:

- Es una Api Open Source desarrollada en JAVA.
- Permite explotar un triplestore RDF ya existente con la tecnología de consulta SPARQL.
- Maneja formatos de datos de nivel de aplicación como JSON y XML
- Amplía el alcance de los datos al incluir para los usuarios el uso de formatos no RDF para acceder a datos.

De acuerdo al análisis previo de PDI descrito en tabla 3.1, no existe un componente que enmarque los requerimientos de la etapa de explotación, Por lo tanto

se justifica como objetivo de esta tesis, desarrollar un componente que permita la explotación del SPARQL del triplestore ya generado en la etapa anterior.

3.2.5.1. Descripción Funcional del Componente para la Explotación de RDF

El componente recibe como entrada una instancia del triplestore generado en la etapa de publicación, a continuación el componente lista las entidades del RDF para que el usuario seleccione las que desea visualizar, así de esta manera dejará únicamente las que considere oportunas a mostrarse en la interfaz web de Elda. De acuerdo a la selección del usuario el componente genera el archivo de configuración de Elda. Finalmente el componente entregará un archivo empaquetado con todos los recursos de la instancia de Elda.

Un esquema funcional del componente se muestra a continuación en la figura 3.6 donde se representa el proceso antes mencionado.



Figura 3.6: Diagrama funcional del componente para la Etapa de Explotación. [3]

3.3. Análisis y Diseño de Patrones de Limpieza

En el proceso de publicación de *Linked Data* contempla un proceso denominado *Cleaning* el cual tiene como objetivo eliminar todos los errores en los datos como sea posible. En la figura 3.7 se muestra algunos posibles errores en los nombres de personas en los repositorios OAI, en este caso se ha definido el formato para este dato como apellidos, nombres. Por lo general, estos datos son ingresados por seres humanos y se equivocan al hacer este trabajo, en consecuencia se debe realizar la limpieza y corrección. En PDI existen múltiples componentes para el manejo de *string*, que facilitan realizar el proceso de *Cleaning* sobre los datos. Por lo cual, no se desarrolla un componente nuevo para realizar este proceso sino que se detectan patrones, los cuales son implementados en PDI para que realicen esta tarea, dependiendo el tipo de error a solucionar.

Datos desde la fuente de datos	Datos realizados la limpieza	Explicación de la limpieza
<u>SantacruzMora, EdisonFreddy</u>	Santacruz Mora, Edison Freddy	Solamente la separación de los apellidos y apellidos
Santacruz M, Edison Freddy	Santacruz M, Edison Freddy	Se lo deja pasar, porque esta correcto aunque solo contenga una siglas
Santacruz, Edison Freddy	Santacruz, Edison Freddy	Se determina que la primera parte es un apellido
Santacruz Mora Edison Freddy, Santacruz Mora Edison Freddy	Santacruz Mora, Edison Freddy	Hay que eliminar los repetidos
Santacruz Mora Edison Freddy, Peñaloza Marín Fabián Leonardo	Santacruz Mora, Edison Freddy Peñaloza Marín, Fabián Leonardo	Hay que separarlos en dos registros diferentes
Santacruz Mora, E	Santacruz Mora, E	Se debería realimentar la base de datos para corregir este tipo errores
Santacruz, Edison	Santacruz, Edison	Se determina que la primera parte es un apellido
Edison, Santacruz	Santacruz, Edison	Se determina que la primera parte es un apellido y si no lo es, se intercambia posiciones

Figura 3.7: Ejemplo de errores en los datos

Al definir un patrón de limpieza para detectar un tipo de error en los datos de un repositorio, puede llegar a ser reutilizado en otros con el mismo tipo de problema como por ejemplo errores por digitación de caracteres de acento, comillas y diéresis los cuales son comunes en muchos repositorios. Entonces la configuración de los componentes de PDI para la corrección de este tipo de error con este patrón sera reutilizable. Ahora bien existen también patrones que detecten errores únicos de cada repositorio debido a peculiaridades propias del mismo, por lo tanto la configuración de los componentes para su corrección no seria reutilizable en otro repositorio. Un esquema funcional de este proceso se muestra en la figura 3.8 que cumplirá el objetivo de detectar el patrón y limpiarlo .



Figura 3.8: Esquema funcional para la limpieza de los datos

La descripción de los patrones para la limpieza de los tipos de errores se detalla en la sección 4.8.

Capítulo 4

Implementación de los Componentes para el Framework Enmarcados en la Metodología para la Publicación de *Linked Data*

UNIVERSIDAD DE CUENCA
desde 1867

4.1. Introducción

En este capítulo se describe la implementación de los componentes analizados en el Capítulo 3, basados en la estructura básica de los componentes en PDI, que se detalla en el anexo 7.1. Para abarcar la implementación de cada componente esta se divide de la siguiente manera:

- Interfaz del componente.
- Funcionalidad e implementación de los campos de la interfaz.
- Implementación de la salida del componente.

4.2. Implementación del Componente para la Extracción de los Datos

Partiendo del análisis realizado en el Capítulo 3 en la sección 3.2.1 para la etapa de especificación es necesario el desarrollo de un componente que permita la extracción de los datos desde repositorios digitales OAIPMH.

4.2.1. Interfaz del Componente *OAILoader*

Los datos de entrada para este componente son extraídos desde un repositorio OAI que almacena información. Primeramente para extraer la información se tiene que conectar a los repositorios OAI a través de una URL, que retorna la información en diferentes formatos con estructura XML tales como: xoi, oai_dc, mets etc... Por lo tanto, este componente permite ingresar una URL, escoger el formato y los campos por el cual se va a recuperar la información. La interfaz del componente tiene un campo Input URL, un botón para recuperar los formatos (Get Formats) y un botón para generar los XPath (Get XPath). Como se muestra en la figura 4.1.

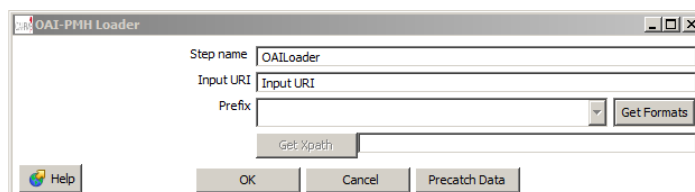


Figura 4.1: Interfaz del componente *OAILoader*

Nota: El campo Step name, el boton OK, Cancel y Help son partes comunes de cualquier componente de PDI. Además, el botón Precatch Data se detalla en la sección 4.9.

4.2.2. Funcionalidad e Implementación de la Interfaz del Componente *OAILoader*

En esta sección se describe la funcionalidad e implementación de los campos con mayor relevancia de la interfaz del componente *OAILoader*.

Campo Input URL

Este campo captura la URL del repositorio digital OAI, para conectarse y extraer los datos. La URL sigue una estructura valida, como se muestra en la figura 4.2 muestra un ejemplo de URL y su respectivo resultado de la invocación.

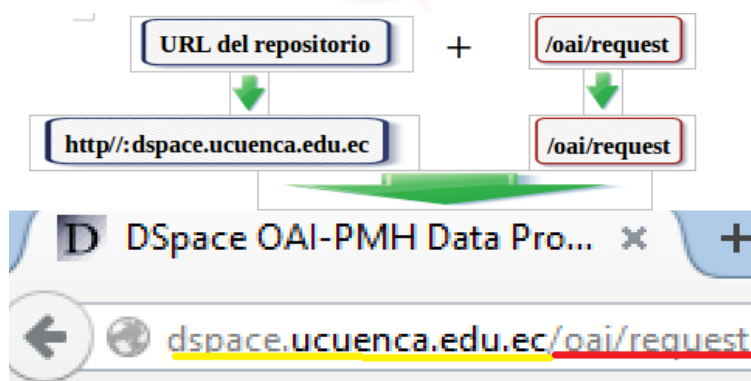


Figura 4.2: Estructura valida de una URL y resultado de la invocación a través de un navegador.

Con la URL capturada, el componente utiliza la librería oai2rdf para conectarse y extraer los datos desde el repositorio. Hay que recalcar, que la URL sola no permite extraer los datos, sino solamente permite conectarse al repositorio. Para extraer los datos se necesita el formato de los datos y el XPath para determinar que datos se obtendrán del repositorio. Se muestra en la figura 4.2 el resultado de conexión al servidor en un navegador.

Esta URL es la base para todo el proceso de explicación de los otros campos del componente.

Botón Get Formats

Esta funcionalidad permite conocer que formatos de datos han sido implementados en el repositorio OAI-PMH, debido a que, un repositorio permite la implementación de alrededor de doce formatos, pero existen instituciones que han implementado solo algunos de ellos. En conclusión esta funcionalidad, ayuda al usuario del componente a seleccionar un formato y evitar que los usuarios envíen como parámetros formatos no implementados en dichos repositorios. Esta funcionalidad en un navegador es embebida en el componente. Como muestra en la figura 4.3.

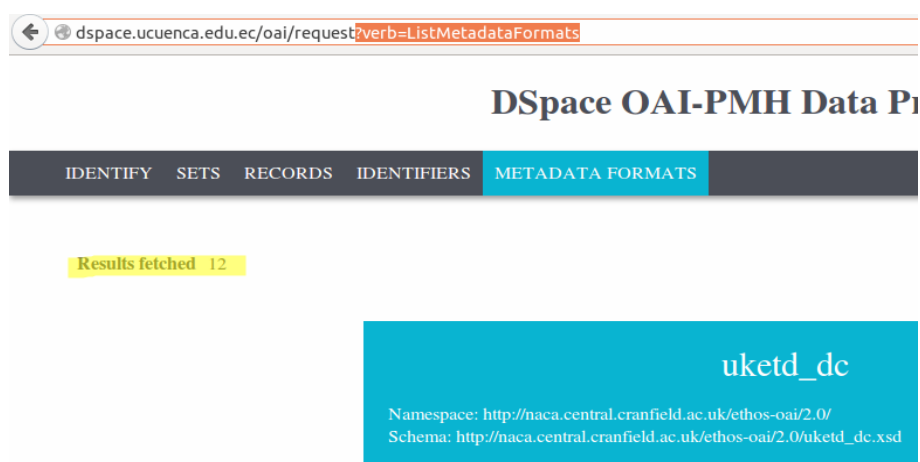


Figura 4.3: Formatos en un navegador web.

El botón Get Formats utiliza la librería oai2rdf para conectarse y extraer los formatos que contiene el repositorio, que finalmente son cargados en la interfaz en un combobox. Hay que hacer notar, que en el combobox existen formatos con el mensaje Not implement yet que significa que no se puede extraer los datos con ese formato, debido a que no están implementados en el componente aunque estén implementados en el repositorio. Ver figura 4.4.

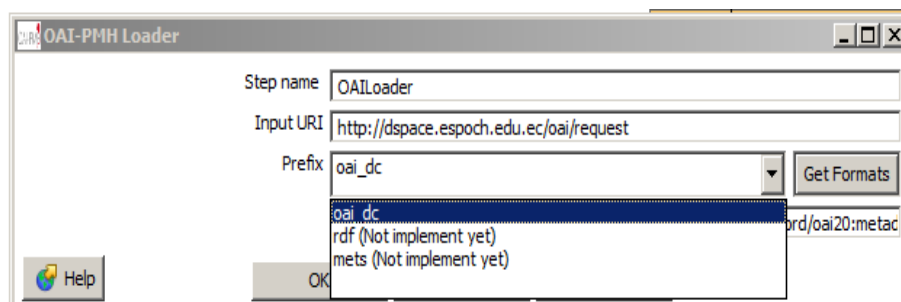


Figura 4.4: ComboBox con los formatos.

Botón Get XPath

Al realizar la extracción de los datos con la librería oai2rdf, se cosecha los registros completos con todos los campos que estos contengan, pero esta funcionalidad no es suficiente cuando se requiere utilizar únicamente un solo campo. En consecuencia, el componente implementa la funcionalidad de escoger todos o un campo específico de los datos de acuerdo a la necesidad del usuario. Así pues, para desarrollar este comportamiento se conoce que los datos están organizados en estructura XML, de la cual se puede obtener los datos mediante un lenguaje de consultas para XML en este caso XPath. Por lo tanto, el objetivo de este botón es la construcción de los XPath de acuerdo al formato elegido. En la figura 4.5 se muestra un extracto de un documento XML.



```
<ListRecords>
  <record>
    <metadata>
      <metadata xmlns="http://www.lyncode.com/xoai">
        <element name="dc">
          <element name="contributor">
            <element name="advisor">
              <element name="es_ES">
                <field name="value">Guerrero Villavicencio, Patricio</field>
              </element>
            </element>
          <element name="author">
            <element name="none">
              <field name="value">Brito Rivas, Mauricio Rodrigo</field>
              <field name="value">Baquero Larriva, Andrés Orlando</field>
            </element>
          </element>
        </element>
      </metadata>
    </record>
  </ListRecords>
```

Figura 4.5: Extracto del un registro con el formato xoai.

La construcción de los XPath se inicia extrayendo la estructura del XML, utilizando la URL y el formato elegido que se envían como parámetros a la librería oai2rdf, la cual retorna los primeros cien registros, pero para construir las rutas XPath se toma el primer registro, como se muestra en la figura 4.5. El objetivo de extraer el primer registro es obtener la estructura del XML de acuerdo al formato elegido, con lo cual, se construye las rutas XPath basado en un algoritmo recursivo que recorre la estructura de árbol del archivo XML. Dicho de otra manera, se construye la rutas XPath recorriendo cada uno de los nodos del árbol XML como se muestra en 4.5 hasta llegar al nodo hoja y en cada cambio de nodo se va almacenando el XPath en una lista que genera como resultado todas las rutas XPath posible. En la figura 4.6 se muestra la lista de XPath obtenidos al elegir el formato xoai.

Selected xpath

```
/oai20:OAI-PMH/oai20:ListRecords/oai20:record/oai20:metadata/xoai:metadata
/oai20:OAI-PMH/oai20:ListRecords/oai20:record/oai20:metadata/xoai:metadata/xoai:element[@name='dc']
/oai20:OAI-PMH/oai20:ListRecords/oai20:record/oai20:metadata/xoai:metadata/xoai:element[@name='dc']/xoai:element[@name='contributor']
/oai20:OAI-PMH/oai20:ListRecords/oai20:record/oai20:metadata/xoai:metadata/xoai:element[@name='dc']/xoai:element[@name='contributor']/xoai:element[@name='advisor']
/oai20:OAI-PMH/oai20:ListRecords/oai20:record/oai20:metadata/xoai:metadata/xoai:element[@name='dc']/xoai:element[@name='contributor']/xoai:element[@name='author']
```

Figura 4.6: XPath obtenidas desde el formato xoai.

4.2.3. Implementación de la Salida del Componente *OAI-Loader*

En el análisis de este componente se determinó que los datos de salida deben ser devueltos en una tabla con tres columnas: Idrecord, Field y Data. Esto se muestra en la figura 4.7.

Id Record	Field	Data
oai:dspace.esPOCH.edu.ec: 123456789/32	identifier	oai:dspace.esPOCH.edu.ec: 123456789/32
oai:dspace.esPOCH.edu.ec: 123456789/32	timestamp	2011-01-20T10:58:20Z
oai:dspace.esPOCH.edu.ec: 123456789/32	setSpec	hdl_123456789_5 => Tesis Ingeniero en Sistemas Informáticos
oai:dspace.esPOCH.edu.ec: 123456789/32	creator	Pinduisaca Esparza, Silvia Patricia
oai:dspace.esPOCH.edu.ec: 123456789/32	creator	Cajas Montero Edison Omar, Cajas Montero Edison Omar
oai:dspace.esPOCH.edu.ec: 123456789/32	date	2010-03-05T16:24:23Z
oai:dspace.esPOCH.edu.ec: 123456789/32	date	2010-03-05T16:24:23Z
oai:dspace.esPOCH.edu.ec: 123456789/32	date	2010-03-05T16:24:23Z
oai:dspace.esPOCH.edu.ec: 123456789/32	identifier	http://hdl.handle.net/123456789/32
oai:dspace.esPOCH.edu.ec: 123456789/32	description	El estudio trata de las características de los paradigmas (modelo
oai:dspace.esPOCH.edu.ec: 123456789/32	language	es
oai:dspace.esPOCH.edu.ec: 123456789/32	relation	UDCTFIYE; 18T00393
oai:dspace.esPOCH.edu.ec: 123456789/32	subject	PROGRAMACION ESTRUCTURADA
oai:dspace.esPOCH.edu.ec: 123456789/32	subject	MODELOS MATEMATICOS
oai:dspace.esPOCH.edu.ec: 123456789/32	subject	PARADIGMAS
oai:dspace.esPOCH.edu.ec: 123456789/32	subject	ESTUDIO COMPARATIVO
oai:dspace.esPOCH.edu.ec: 123456789/32	subject	PLANIFICACION DE LA PRODUCCION
oai:dspace.esPOCH.edu.ec: 123456789/32	subject	PLANTA DE LACTEOS ESPOCH
oai:dspace.esPOCH.edu.ec: 123456789/32	title	Estudio Comparativo de los Paradigmas de Programación Aplicado
oai:dspace.esPOCH.edu.ec: 123456789/32	type	Thesis

Figura 4.7: Salida del componente *OAI-Loader*

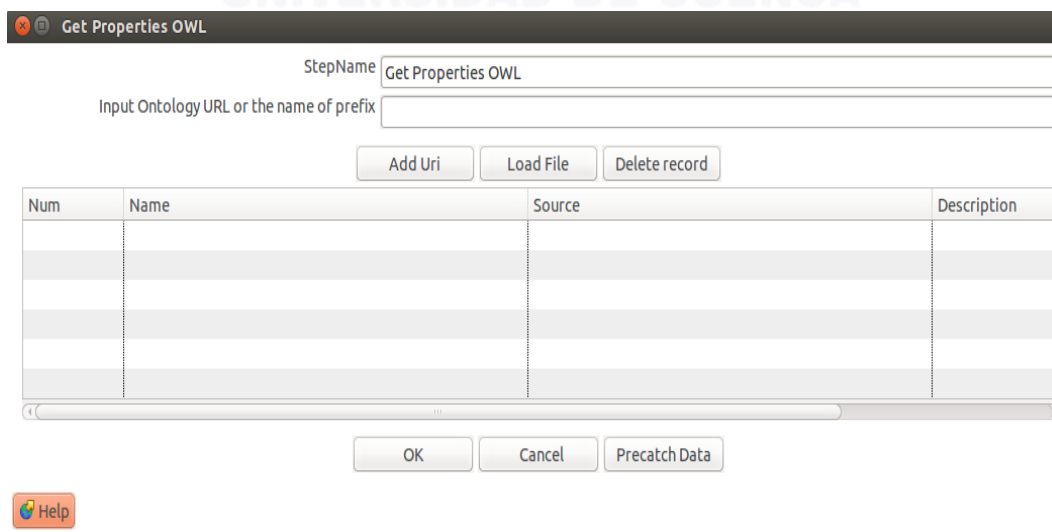
Para comenzar la extracción de los datos el componente utiliza todos los campos configurados en la interfaz del componente (URL, formato, XPath) que son enviados como parámetros a la librería oai2rdf. Como se mencionó anteriormente los registros están contenidos en archivos XML, el cual se accede a través de una función recursiva que va recorriendo cada nodo del árbol XML. La información obtenida en el proceso recursivo es la cabecera que tiene cada registro y los campos especificados en el XPath como se muestra en la figura 4.7.

4.3. Implementación del Componente para la Carga de las Ontologías

De acuerdo al análisis del capítulo 3 en la sección 3.2.2, en la etapa de modelado se determinó la necesidad de desarrollar un componente que permita la carga de propiedades y clases de ontologías, con el fin de seleccionar los vocabularios adecuados para el dominio de los datos.

4.3.1. Interfaz del Componente *Get Properties OWL*

El componente recibe como entrada una o varias ontologías, donde por cada una de ellas se listarán sus propiedades y clases. Para agregar cada ontología se permite ingresar una URL o seleccionar un archivo local de la ontología, esto determina que la interfaz del componente tenga un campo Input URI or name of prefixlos botones Add URI, Load File, Delete Record y una tabla donde se listan las ontologías que han sido ingresadas. Ver figura 4.8.



Num	Name	Source	Description

Figura 4.8: Interfaz del componente *Get Properties OWL*

Nota: El botón Precatch Data se describe, su implementación y funcionalidad en la sección implementaciones comunes [4.9.1.2].

4.3.2. Funcionalidad e Implementación de la Interfaz del Componente *Get Properties OWL*

A continuación se describe la funcionalidad e implementación de cada uno de los campos que componen la interfaz del componente *Get Properties OWL*.

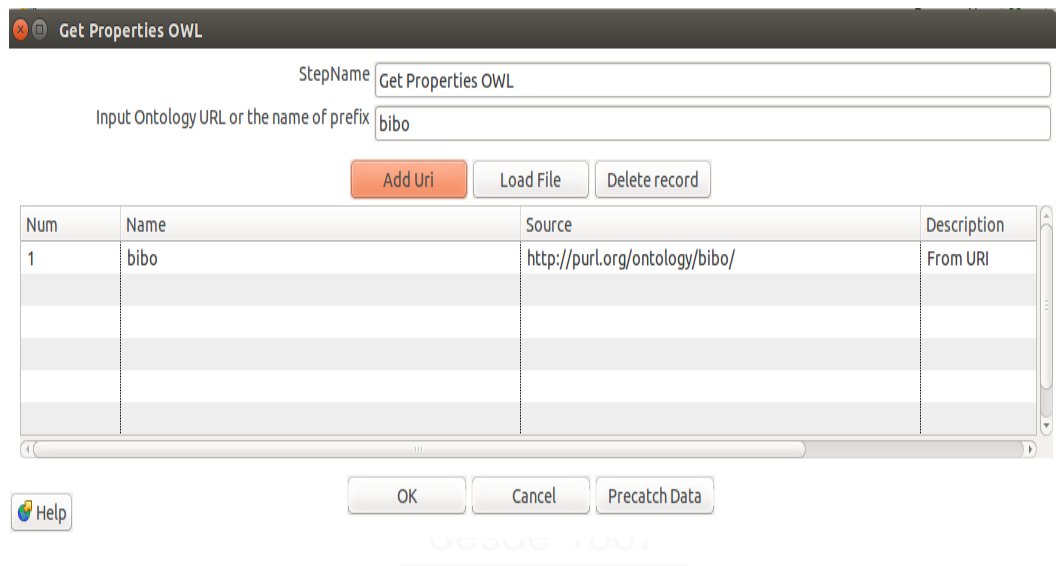


Figura 4.9: Boton Add URI al encontrar un namePrefix

Campo Input URI or name of prefix

En este campo se visualiza el origen de la ontología que se va ingresar, este valor puede ser de dos maneras: La primera opción es una ruta local del archivo de la ontología, este valor se cargará automáticamente cuando se accione el botón Load File: la segunda opción es mediante una URL de la ontología en la web, la cual se cargará automáticamente con la acción del botón Add URI.

Botón Add URI

El botón Add URI agrega una ontología en la tabla de la interfaz tomando como entrada el valor del campo Input URI or name of prefix. Para facilidad del usuario se permite ingresar el prefixName, el cual es procesado por el componente automáticamente para encontrar la URI respectiva a través de la consulta del servicio <http://lov.okfn.org/dataset/lov/api/v2/vocabulary/search>. Dicho de otra manera el usuario podría ingresar el prefixName bibo y el componente encontrara la URI <http://purl.org/ontology/bibo/> que es la que corresponde. Se debe acotar que se puede ingresar la URI completa en el campo de texto Input

URI or name of prefix y el componente también la procesará normalmente.

Además se realizan validaciones en el caso de no encontrar ninguna URL correspondiente a la búsqueda, en consecuencia el componente muestra un mensaje al usuario de URL no encontrada.

En la implementación del botón Add URI se toma como parámetro el valor ingresado en el campo Input URI or name of prefix como prefixName, después a través del servicio <http://lov.okfn.org/dataset/lov/api/v2/vocabulary/search> se obtiene la URL respectiva de la ontología.

Botón Load File

Al dar clic en este botón se despliega un dialogo que permite seleccionar un archivo local de la ontología que se desea ingresar. El valor de la ruta y su nombre se agregan en nueva fila de la tabla de ontologías. Esto se muestra en 4.10.

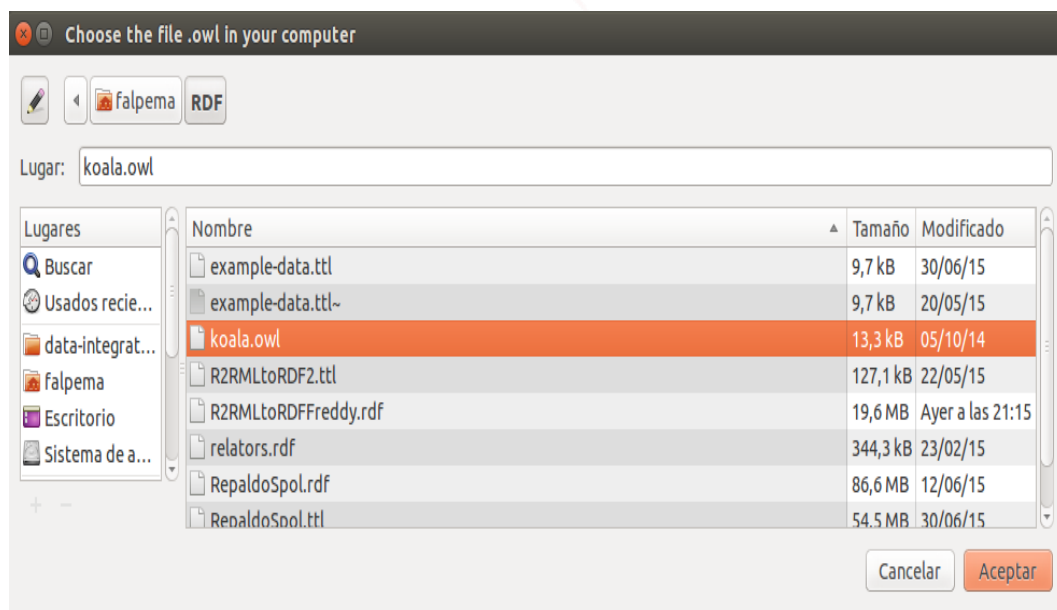


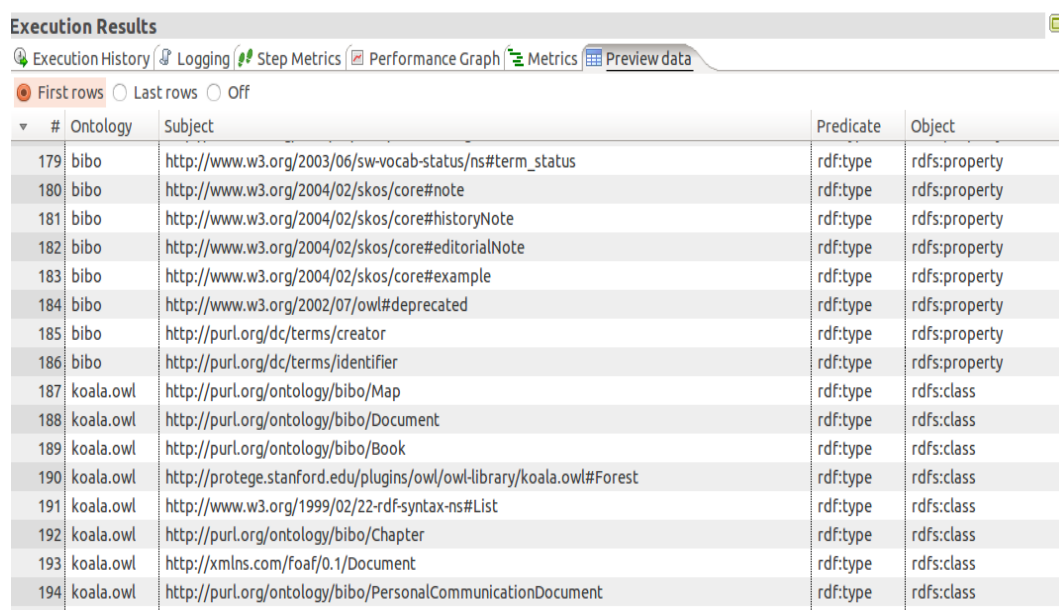
Figura 4.10: Dialog para seleccionar un ontología desde un archivo local.

Botón Delete Record

Este campo permite borrar una ontología seleccionada de la tabla de ontologías.

4.3.3. Implementación de la Salida del Componente *Get Properties OWL*

En el análisis de este componente se determinó que los datos de salida serán listados en una tabla con cuatro columnas: Num, Name, Source, Descripción para facilitar el manejo de los vocabularios dentro de PDI. Como se muestra en la figura 4.11



The screenshot shows a web application interface with a tabbed menu at the top: 'Execution History', 'Logging', 'Step Metrics', 'Performance Graph', 'Metrics', and 'Preview data'. The 'Preview data' tab is active, displaying a table of execution results. The table has five columns: '#', 'Ontology', 'Subject', 'Predicate', and 'Object'. The data rows show various RDF properties from different ontologies, including 'bibo' and 'koala.owl'. The 'Subject' column contains URIs for specific terms, and the 'Object' column shows the corresponding RDF type or class.

#	Ontology	Subject	Predicate	Object
179	bibo	http://www.w3.org/2003/06/sw-vocab-status/ns#term_status	rdf:type	rdfs:property
180	bibo	http://www.w3.org/2004/02/skos/core#note	rdf:type	rdfs:property
181	bibo	http://www.w3.org/2004/02/skos/core#historyNote	rdf:type	rdfs:property
182	bibo	http://www.w3.org/2004/02/skos/core#editorialNote	rdf:type	rdfs:property
183	bibo	http://www.w3.org/2004/02/skos/core#example	rdf:type	rdfs:property
184	bibo	http://www.w3.org/2002/07/owl#deprecated	rdf:type	rdfs:property
185	bibo	http://purl.org/dc/terms/creator	rdf:type	rdfs:property
186	bibo	http://purl.org/dc/terms/identifier	rdf:type	rdfs:property
187	koala.owl	http://purl.org/ontology/bibo/Map	rdf:type	rdfs:class
188	koala.owl	http://purl.org/ontology/bibo/Document	rdf:type	rdfs:class
189	koala.owl	http://purl.org/ontology/bibo/Book	rdf:type	rdfs:class
190	koala.owl	http://protege.stanford.edu/plugins/owl/owl-library/koala.owl#Forest	rdf:type	rdfs:class
191	koala.owl	http://www.w3.org/1999/02/22-rdf-syntax-ns#List	rdf:type	rdfs:class
192	koala.owl	http://purl.org/ontology/bibo/Chapter	rdf:type	rdfs:class
193	koala.owl	http://xmlns.com/foaf/0.1/Document	rdf:type	rdfs:class
194	koala.owl	http://purl.org/ontology/bibo/PersonalCommunicationDocument	rdf:type	rdfs:class

Figura 4.11: Salida del componente *Get Properties OWL*.

Para comenzar la extracción de las clases y propiedades, el componente utiliza todas las rutas de ontologías ingresadas por el usuario en la tabla de la interfaz. Estas rutas locales o URLs de ontologías son enviadas como parámetros a la librería Jena 2.5.4.1. Es decir por cada ontología mediante Jena se crea un modelo *OntoModel* en donde se carga su estructura completa (clases y propiedades). Una vez validada la carga correcta del modelo se procede a acceder su estructura mediante un algoritmo de recorrido, es decir por cada iteración del algoritmo se extrae cada clase y propiedad para ser almacenada en la tabla de salida. Finalmente cabe recalcar que cada registro se ingresa en la base de datos embebida para poderse cargar en el resto de componentes mediante la función de precarga.

4.4. Funcionalidad del Componente *Ontology Mapping*

En esta sección se describe la funcionalidad del componente *Ontology Mapping*, el cual pertenece a la etapa de modelado de la metodología LOD, donde es necesario disponer de un componente que permita realizar el mapeo entre los datos y las ontologías. Hay que recalcar que este componente no fue desarrollado en el presente trabajo de tesis por lo tanto solo se describirá su funcionalidad.

4.4.1. Interfaz del Componente *Ontology Mapping*

El componente recibe dos entradas: la primera es la fuente de Datos del repositorio proporcionado por el componente *OAILoader*, la segunda son las ontologías configuradas en el componente *Get Properties OWL*. En la figura 4.12 se muestra la interfaz del componente *Ontology Mapping*.

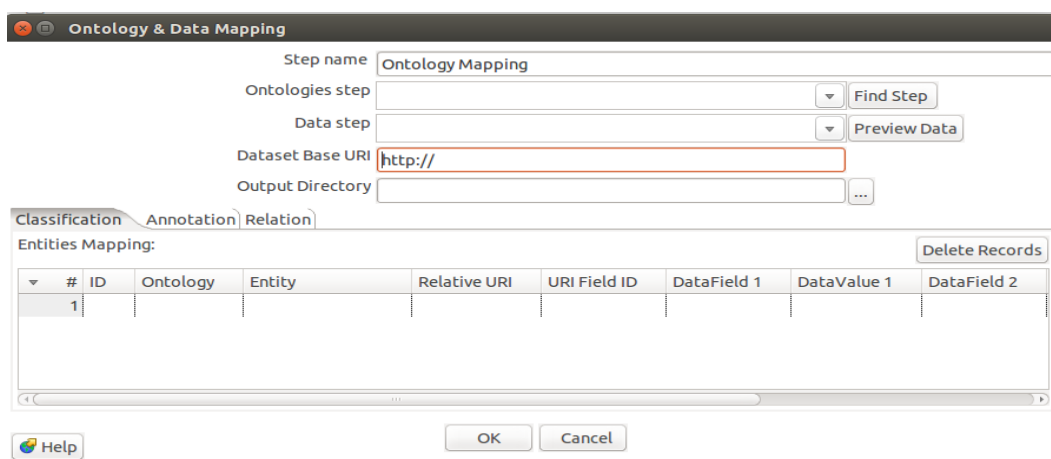


Figura 4.12: Interfaz del componente *Ontology Mapping*

Los primeros parámetros que se configuran como base de este componente son los siguientes:

- StepName: Es el nombre que se asigna al componente para la transformación.
- Ontologies Step: Combobox para seleccionar un componente de tipo *Get Properties OWL* del cual se cargan las ontologías.

- Data step: Combobox para seleccionar un componente ***OAILoader*** del cual se cargan los datos.
- Dataset Base URI: Campo de texto para asignar la base URI con la cual se construirá la URL de los recursos generados.
- Output Directory: Campo de texto donde se visualiza el directorio seleccionado para el archivo de mapeo en lenguaje R2RML.

El componente tiene 3 pestañas donde se realiza la configuración para el mapeo, como son: *Classification*, *Annotation* y *Relation*. La explicación de la configuración de las pestañas para realizar el mapeo se describe en la sección 5.4.2.2.

4.4.2. Salida del Componente *Ontology Mapping*

La salida del componente es el mapeo entre ontologías y datos, para lo cual se genera un archivo en el lenguaje de mapeo r2rml. Finalmente este archivo de mapeo será la entrada para el componente *R2RMLtoRDF* de la etapa de generación.

4.5. Implementación del Componente para la Generación de Datos en el Estándar RDF

La sección 3.2.3, detalla la etapa de generación, la cual determinó la necesidad de desarrollar un componente que permita la generación de datos en el estándar RDF y que la salida de este, sea un archivo con las tripletas RDF de los datos del repositorio configurado en el componente ***OAILoader***.

4.5.1. Interfaz del Componente *R2RMLtoRDF*

Este componente hace uso de la librería db2triples de la sección 2.5.3, la cual necesita como entrada un archivo de mapeo entre los datos y ontologías, además una conexión a base de datos para generar los datos en formato RDF. Hay que recalcar, que este proyecto plantea unificar las tecnologías utilizadas para generar RDF, por lo cual, se utiliza una base de datos embebida compatible con el lenguaje JAVA como lo es H2, ver sección 4.9.1. En consecuencia el componente permite escoger el archivo de mapeo, configurar los parámetros de conexión a la base de datos, seleccionar el formato en el cual se estructuran las tripletas RDF. Como

se muestra en la figura 4.13, al mismo tiempo permite escoger el directorio para el archivo de datos RDF de salida y utiliza dos botones adicionales para facilidad en el uso del componente.

Nota.- El nombre del archivo de salida se toma desde el campo StepName.

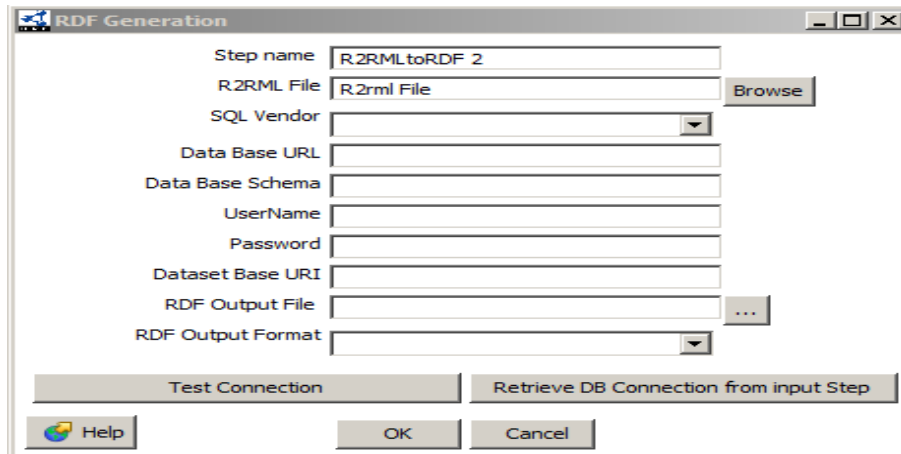


Figura 4.13: Interfa del componente *R2RMLtoRDF*

4.5.2. Funcionalidad e Implementación de la Interfaz del Componente *R2RMLtoRDF*

En esta subsección se describe la funcionalidad e implementación de los campos mas relevantes que componen la interfaz del componente *R2RMLtoRDF*.

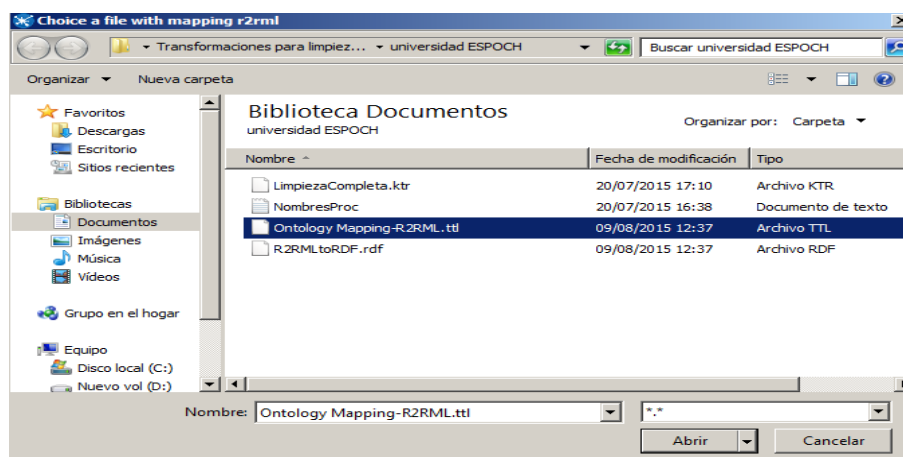


Figura 4.14: FileDialog para elegir el archivo de mapeo.

Campo R2RML file

La funcionalidad de este campo es registrar la ruta del archivo de mapeo. La implementación se la realiza con un `FileDialog` de la librería `swt`, que permite escoger del archivo de mapeo guardado en la computadora. Una vez seleccionado el archivo, se utiliza la librería `db2triples` para extraer el mapeo entre las ontologías y los datos. Como se muestra en la figura 4.14.

Campos de Configuración de la Conexión a Base de Datos

El objetivo de cada uno de estos campos es registrar información que permita la conexión a base de datos, creada por el componente ***OAILoader*** en la sección 5.2. Cada campo se detalla a continuación:

- **SQLVendor**: Permite elegir el driver de la base de datos.
- **Data Base URL**: Dirección URL de conexión a la base de datos.
- **Data Base Schema**: Nombre de la base de datos.
- **UserName**: Nombre de usuario de la base de datos.
- **Password**: Contraseña de la base de datos.

La implementación de la conexión a la base de datos, se realiza con la librería `db2triples`, la cual permite conectar a diferentes gestores de bases de datos, tales como: `Mysql`, `PostgreSQL` y `H2`, por lo cual, esta toma todos los parámetros configurados en la interfaz del componente para realizar la conexión.

RDF Output File

En este campo se almacena el directorio en el cual se crea el archivo de salida con el formato.

Botón Test Connection

Este botón realiza una conexión a la base de datos, para comprobar que los parámetros configurados sean correctos, además da acceso a los datos antes de ejecutar el componente.

Botón Retrieve DB Connection from input Step

El propósito de este botón es configurar automáticamente todos los parámetros de conexión a base de datos y la ruta al archivo de mapeo, esto se consigue cuando algún componente esta conectado como entrada al componente ***R2RMLtoRDF***,

y de este consigue todas las configuraciones previas realizadas, permitiendo integrar los componentes previamente configurados. Finalmente, la implementación de esta funcionalidad es para facilitar el uso del componente por parte del usuario.

4.5.3. Implementación de la Salida del Componente *R2RMLtoRDF*

La implementación de la salida del componente se la realiza con la librería *db2triples*, la cual toma el archivo de mapeo y la conexión a base de datos como entradas, con los cuales extraer las relaciones creadas en el archivo de mapeo y con los registros desde la base de datos, realiza la generación de RDF y crea el archivo en la ruta especificada. Como se muestra en la figura, la salida en PDI es la ruta en donde se almaceno el archivo creado.

4.6. Implementación del Componente para la Publicación de los Datos

De acuerdo al análisis del capítulo 3 realizado para la etapa de publicación se determinó desarrollar un componente que permita realizar la publicación del RDF obtenido en la etapa generación mediante el Servidor Fuseki, el objetivo es crear una instancia de Fuseki configurado automáticamente para generar un servicio de SPARQL Endpoint del triplestore con el RDF de la etapa de generación.

4.6.1. Interfaz del Componente *Fuseki Loader*

Este componente recibe como entrada una archivo RDF para ser publicado en triplestore de Fuseki. Por lo tanto esta necesidad determina que la interfaz del componente tenga los campos Input Dataset, Service Name, Service Port, Base URI que son necesarios para la configuración de Fuseki. Finalmente se muestra en la figura 4.19 la interfaz descrita.

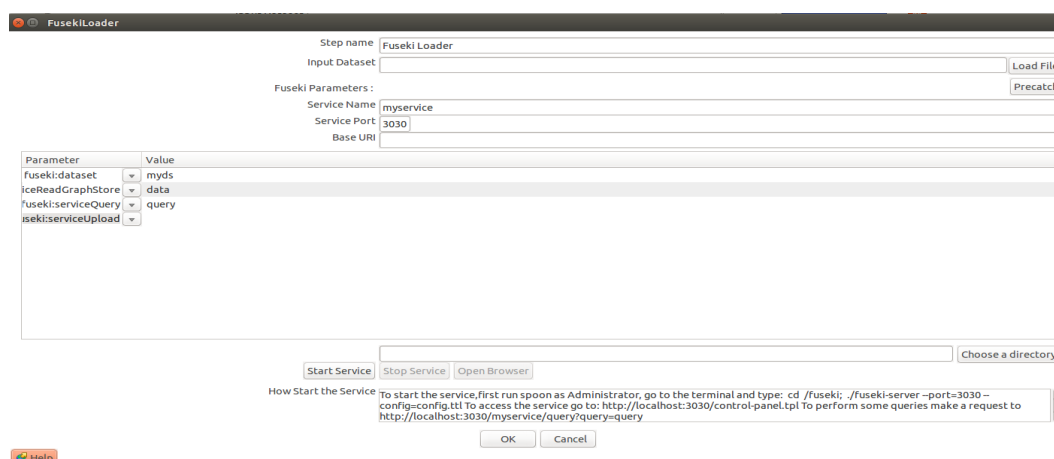


Figura 4.15: Interfaz del componente **Fuseki Loader**

Los parámetros de configuración son.

- **StepName:** Campo de texto donde se ingresa el nombre que se asignara al componente en la transformación.
- **Input Dataset:** Campo donde se selecciona el archivo RDF que será utilizado como Dataset. Nota: por medio del botón Precatch se carga automáticamente este valor del componente **R2RMLtoRDF**.
- **Service Port:** Campo de texto donde se ingresa el puerto por donde se inicia el servicio de Fuseki.
- **Base URI:** Campo de texto para asignar el grafo con el que se creará el servicio (BaseURI).
- **Choose Directory:** Campo de texto que visualiza el directorio seleccionado en donde se creará la instancia de Fuseki.

Ahora bien Fuseki permite configurar varias propiedades para generar un servicio de SPARQL Endpoint como se puede consultar en su documentación oficial [9]. Con respecto a esta característica fue necesario crear una tabla de parametrización en donde por cada fila, se tenga precargado un combobox con las propiedades soportadas por Fuseki. Por cada fila se selecciona una propiedad y se asigna un valor para la misma mediante un campo texto ubicado a la derecha de dicha propiedad. Para facilidad del usuario el componente permite desde la interfaz iniciar y detener el servicio, además si el servicio esta levantado se puede abrir un navegador web que estará apuntando a la URL del servicio. Esta

funcionalidad se implementa con los botones Start Service, Stop Service y Open Browser.

4.6.2. Funcionalidad e Implementación de la Interfaz del Componente Fuseki Loader

En esta sección se describe la funcionalidad e implementación de cada uno de los campos que componen la interfaz del componente **Fuseki Loader**.

Campo de Texto Input Dataset

Este campo especifica la ruta del archivo RDF que servirá como Dataset para la instancia de Fuseki. Su valor se puede tomar mediante el botón LoadFile, o por medio del botón Precatch el cual extrae la ruta del componente anterior **R2RMLtoRDF2**.

Botón Load File

La funcionalidad de este botón es desplegar un cuadro de dialogo que permite seleccionar un archivo local del RDF que se desea ingresar. El valor de la ruta se asigna en el campo de texto Input Dataset. Como se muestra en la figura 4.16.

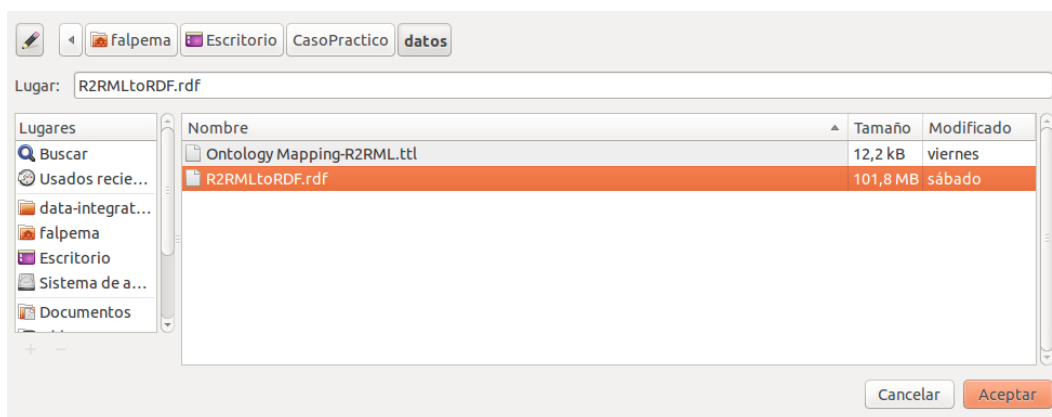


Figura 4.16: Dialogo para seleccionar el RDF desde un archivo local.

La implementación de este botón se realiza mediante la librería SWT para la creación de un cuadro de dialogo que permita seleccionar los archivos.

Tabla Parameter Value

En esta tabla al dar clic en el combobox de parámetros se listarán las propiedades de configuración soportadas por Fuseki. Cabe recalcar que solo la columna Value

es editable para ingresar el valor de la propiedad. Como se muestra en la figura 4.17.

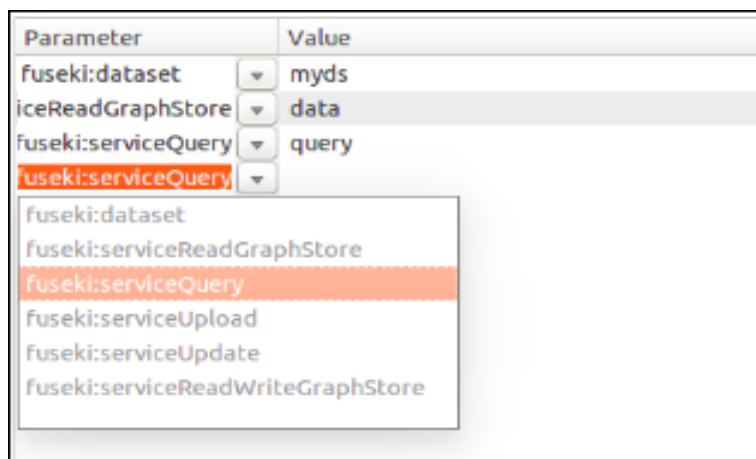


Figura 4.17: Dialogo para seleccionar un rdf desde un archivo local.

Cada fila parametrizada en la tabla será agregada al archivo de configuración de Fuseki que genera el componente.

La implementación de este tabla dinámica se realiza mediante el uso la clase Table de librería SWT de Java. Por medio del uso de un Listener, este crea una nueva fila automáticamente cuando se seleccione la ultima celda de la columna Value.

Botón Start Service

Este botón solo se habilitará cuando el componente haya sido ejecutado al menos una vez, puesto que esto genera una instancia de Fuseki. Dicho de otra manera no se podría iniciar una instancia que aun no ha sido creada.

Al dar clic en este botón se crea un proceso que arranca el servidor Fuseki. Cabe recalcar que el proceso se realiza sobre el directorio que se escogió para la instancia de Fuseki.

La implementación de este botón se realiza mediante Threads para optimizar la eficiencia del componente en tiempo de ejecución. Con el uso de la librería Runtime se manda ejecutar el comando de Fuseki que inicia el servicio, `./Fuseki-server -port=8080 -config=config.ttl` con la parametrización del usuario. Finalmente mediante la clase `ExecutorTask` se asigna un thread al proceso de tipo Runtime que levanta el servicio.

Botón Stop Service

Este botón se habilita cuando el servicio de Fuseki está levantado. Finalmente este botón sirve para detener el servicio, puesto que se tiene identificado la instancia del proceso que levantó el servicio.

Botón Open Browser

La funcionalidad de este botón permite abrir un navegador que accede a la URL del servicio de Fuseki que se ha creado. De esta manera se facilita al usuario el acceso para que pueda realizar la explotación del triplestore mediante consultas SPARQL. Cabe recalcar que el botón estará habilitado solo cuando el servidor Fuseki este arrancado. Ver imagen 4.18.

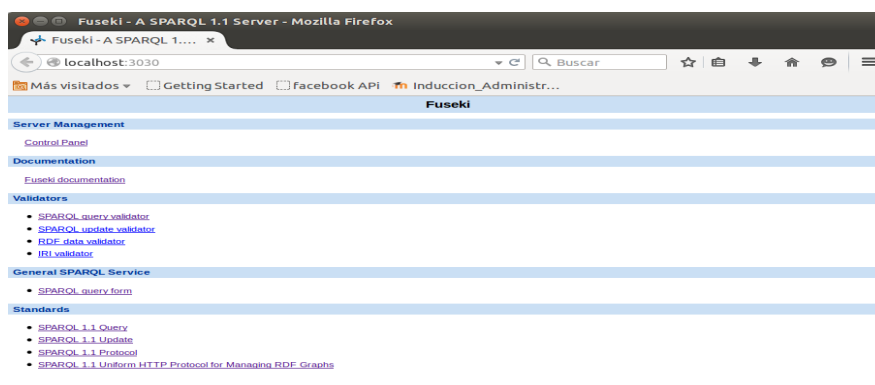


Figura 4.18: Servicio levantado en Fuseki Server

4.6.3. Implementación de la Salida del Componente Fuseki Loader

La salida del componente es una instancia del servidor Fuseki con la configuración parametrizada por el usuario en la interfaz del componente. Para comenzar la creación del archivo de configuración el componente utiliza todos los campos configurados en la interfaz del componente (Nombre del Servicio, Puerto, Dataset, GraphUri, etc) que son enviados como parámetros a la librería Jena. Con el uso de Jena se crea un Rdf Model en el cual se agregan los recursos y propiedades que mapean la configuración descrita en los parámetros. Para ello se crearon clases para definir las propiedades de Fuseki que se desean mapear en el modelo. Una vez creado el modelo con la configuración deseada se procede a escribir en el archivo config.ttl con sintaxis Turtle 2.2.4.2. Cabe recalcar que todos los archivos del servidor Fuseki necesarios para generar su instancia se encuentran embebidos dentro de los recursos del componente. Finalmente el componente coloca el archivo de configuración y el archivo RDF (Dataset) en sus directorios respectivos

dentro de la instancia de Fuseki que genera el componente.

4.7. Implementación del Componente para la Explotación de los Datos

De acuerdo al análisis del capítulo 3 sección 3.2.4, en la etapa de explotación se determinó desarrollar un componente que permita realizar la explotación del SPARQL Endpoint obtenido en la etapa de publicación mediante el Servidor Elda 2.5.6.

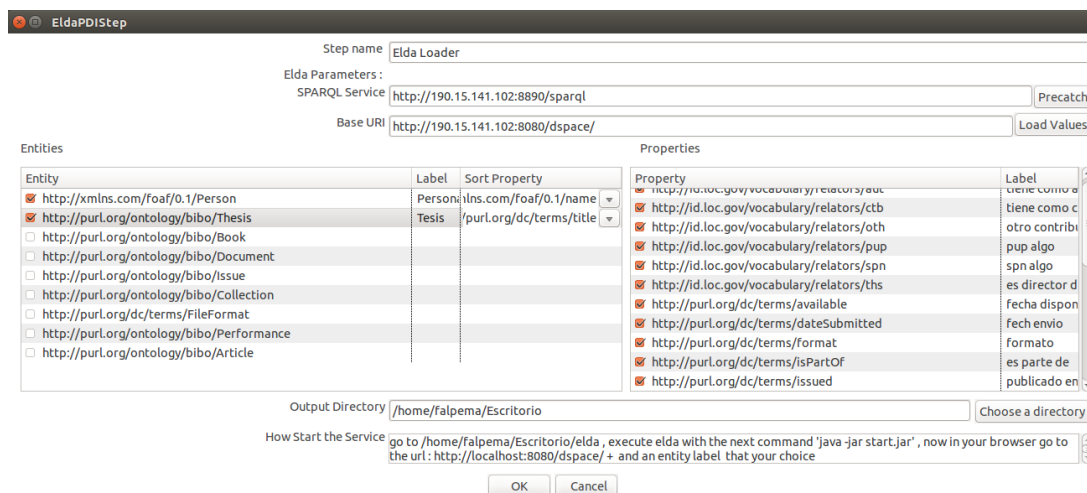
4.7.1. Interfaz del Componente Elda Step

Este componente recibe como entrada un servicio SPARQL EndPoint que consuma el triplestore generado en la etapa de publicación. Primeramente en la configuración de Elda se debe definir las Entidades que mostrará la aplicación. Por cada entidad se pueden configurar varias propiedades a mostrarse, cabe recalcar que en cada propiedad se puede asignar un label específico. Además Elda permite en cada entidad asignar una propiedad por la cual ordenar los resultados en la lista de recursos que se crea. Los campos para esta interfaz son:

- StepName: Campo de texto donde se ingresa el nombre que se asignará al componente en la transformación.
- SPARQL Service: Campo donde se ingresa la URL del servicio SPARQL a consultar. Nota : por medio del botón Precatch se carga automáticamente este valor del componente **Fuseki Loader**.
- Base URI: Campo de texto para ingresar el grafo por el cual se realizaran las consultas en el servicio.
- Output Directory: Es el directorio en donde se generará las instancias de Elda con todos sus archivos configurados.

Con el fin de brindar una interfaz intuitiva para el usuario, se creó la tabla Entities solo para la configuración de la entidades y otra tabla solo para la configuración de propiedades llamada Properties.

Como se muestra en la figura 4.19 la interfaz descrita.



The screenshot shows the 'Elda Loader' window. It has a 'Step name' field set to 'Elda Loader'. Below it, 'Elda Parameters' includes a 'SPARQL Service' field with the URL 'http://190.15.141.102:8890/sparql' and a 'Precatch' button. The 'Base URI' field contains 'http://190.15.141.102:8080/dspace/' with a 'Load Values' button. There are two tables: 'Entities' and 'Properties'. The 'Entities' table has columns for Entity, Label, and Sort Property, with several rows of URIs and labels like 'Person' and 'Tesis'. The 'Properties' table has columns for Property and Label, with rows of URIs and labels like 'tiene como co' and 'otro contri'. At the bottom, there is an 'Output Directory' field set to '/home/falpema/Escritorio' and a 'Choose a directory' button. A text box explains how to start the service, and there are 'OK' and 'Cancel' buttons.

Figura 4.19: Interfaz del componente **Elda Loader**

Nota: El campo Step name, el botón OK, Cancel y Help son parte de cualquier componente de PDI. Además El botón Precatch Data describe su implementación y funcionalidad en la sección implementaciones comunes.

4.7.2. Funcionalidad e Implementación de los Campos de la Interfaz del Componente Elda Step

En esta sección se describe la funcionalidad e implementación de los campos con mayor relevancia de la interfaz del componente **Elda Step**.

Botón Load Values

Su funcionalidad consiste en obtener las entidades existentes en el SPARQL Endpoint mediante una consulta SPARQL, los resultados obtenidos en la consulta se proceden a cargar en la tabla de Entidades.

El botón Load Values, utiliza la clase QueryFactory de la librería Jena para conectarse al SPARQL Endpoint y extraer los resultados de la consulta. Cabe recalcar que en la tabla se cargarán todas las entidades pertenecientes al grafo ingresado por el usuario.

Tabla Entities

La funcionalidad de la tabla es crear un lista de recursos que pertenezcan a la entidad que se seleccione. Por lo tanto, en cada entidad seleccionada hay 3 columnas de tipo checkbox, textbox y combobox respectivamente. La funcionalidad

de cada columna se detalla a continuación.

- **Columna Entity:** Cuando se activa el checkbox el componente procede a ejecutar una consulta SPARQL para obtener las propiedades pertenecientes a la entidad seleccionada, después se cargan los resultados en la tabla Properties y en el comboBox Sort Property.
- **Columna Label:** Este campo de texto permite asignar una etiqueta a la entidad. Además el label también servirá para crear la URL que acceda a la lista de recursos de la entidad seleccionada. Por ejemplo se muestra en la figura 4.20, para la entidad `http://purl.org/ontology/bibo/Thesis` se configura el label con Tesis.

Base URI

Entities

Entity	Label	Sort Property
<input checked="" type="checkbox"/> <code>http://xmlns.com/foaf/0.1/Person</code>	Persona	oaf/0.1/firstName
<input checked="" type="checkbox"/> <code>http://purl.org/ontology/bibo/Thesis</code>	Tesis	org/dc/terms/title

Figura 4.20: Ejemplo de configuración de label para la entidad Thesis

Entonces la URL que se creará para la lista de recursos de este tipo será `http://localhost:8080/dspace/Tesis` como se muestra en la figura 4.21.

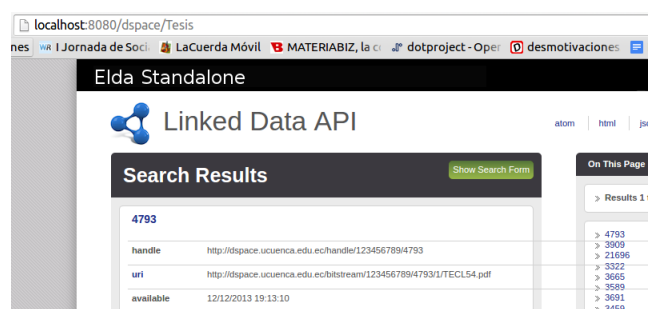


Figura 4.21: Ejemplo de URL formada para la lista de recursos de la entidad Tesis.

- **Columna Sort Property:** La funcionalidad del comboBox Sort Property es permitir elegir la propiedad por la cual se ordenaran la lista de recursos. Hay que hacer notar que este comboBox solo estará cargado mientras se encuentre activo el checkbox de la primera columna.

Tabla Properties

Esta tabla permite seleccionar y configurar propiedades por cada entidad seleccionada en la tabla Entities.

La tabla tiene dos columnas tipo checkbox y textbox respectivamente. La funcionalidad de cada columna se describe a continuación.

- Columna Property: Su funcionalidad permite seleccionar las propiedades que se van a configurar para la entidad.
- Columna Label: Esta campo captura el valor que se asignará como etiqueta a la propiedad.

En conclusión la funcionalidad de esta tabla permite personalizar los nombres con los cuales se visualizarán las propiedades de cada recurso encontrado dentro de Elda.

4.7.3. Implementación de la Salida del Componente Elda Step

La salida del componente es una instancia del servidor Elda con la configuración parametrizada por el usuario en la interfaz del componente. Para comenzar la creación del archivo de configuración el componente utiliza todos los campos configurados en la interfaz del componente (Entidades Seleccionadas, Propiedades Seleccionadas, Labels, Sort Property) que son enviados como parámetros a la librería Jena. Con el uso de Jena se crea un RDF Model en el cual se agregan los recursos y propiedades que mapean la configuración descrita en los parámetros. Para ello se crearon clases para definir las propiedades de Elda que se desean mapear en el modelo. Una vez creado el modelo con la configuración deseada se procede a escribir en un archivo con sintaxis Turtle (TTL). Cabe recalcar que todos los archivos del servidor Elda necesarios para generar su instancia se encuentran embebidos dentro de los recursos del componente. Finalmente el componente coloca el archivo de configuración en su directorio respectivo para la instancia generada.

4.8. Implementación de los Patrones de Limpieza

Como se indicó en el capítulo 3 en la sección 3.3, no se desarrollo un nuevo componente, sino se han detectado patrones para la limpieza de los datos, los cuales han sido clasificados como: reutilizables y no reutilizables en cualquier repositorio. Además, en esta sección se detalla algunos aspectos que se deben tener en cuenta como se crearon los patrones, tales como: código de colores, notas.

4.8.1. Patrón Reutilizable

Este tipo de patrón se reutiliza sobre datos de cualquier repositorio. Un patrón reutilizable puede comprobar una estructura definida filtrando todos los registros que la cumplan, un ejemplo de estructura es Apellido1 Apellido2, Nombre1 Nombre2, hay que aclarar que las estructuras las define el desarrollador, entonces todos los datos que ingresen al patrón y cumplan esta estructura serán validos. Para ejemplificar si ingresan al patrón 2 registros: registro1= Santacruz Mora, Edison Freddy y registro2= Santacruz Mora, Edison este determina como salida que el registro1 es correcto y el registro2 es incorrecto. La creación de algunos patrones reutilizables se los describe en la sección 5.2.2.

4.8.2. Patrón no Reutilizable

Este tipo de patrón solamente corrigen errores únicos de un repositorio o anteriormente no detectados en otros repositorios, este tipo de patrones pueden llegar a ser reutilizables cuando se examine datos de otros repositorios, un ejemplo de patrón no reutilizable puede comprobar la valides de los datos que no han cumplido la estructura definida para el patrón reutilizable, pero ahora estos datos tienen una nueva estructura Apellido1 Apellido2, Nombre1 que sigue siendo valida, entonces todos los datos que ingresen al patrón se filtra la nueva estructura valida. En el ejemplo para un patrón reutilizable se comprobó que el registro1= Santacruz Mora, Edison Freddy es valido y que el registro2= Santacruz Mora, Edison es incorrecto, entonces el registro2 sirve como entrada al patrón no reutilizable que comprueba la nueva estructura y determina que el registro2 es valido. La creación de algunos patrones no reutilizables se los describen en la sección 5.2.2.

4.8.4. Notas

En las notas se detalla las correcciones realizadas sobre los datos, en cada bloque, como se muestra en la figura 4.23.

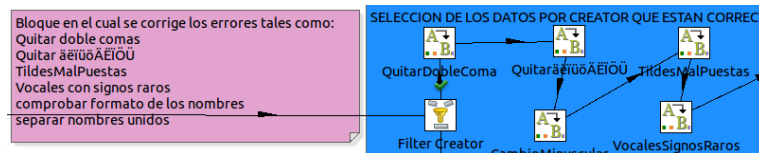


Figura 4.23: Detalle de las correcciones realizadas en cada bloque.

4.9. Implementación de la Funcionalidad Utilitaria para los Componentes

La finalidad de esta sección es explicar como un proceso implementado en el Framework colabora con la integración de los componentes del Framework, el cual se detallara a continuación.

4.9.1. Inclusión de Base de Datos H2

Este proceso realiza la inclusión de una base de datos embebida H2 [2.4.2], la cual es integrada en los componentes desarrollados en este proyecto, por ser compatible con el lenguaje de programación JAVA. El propósito de este proceso es permitir el acceso a los datos extraídos en la etapa de especificación, como también a las clases y propiedades de las ontologías en la etapa de modelado entre los demás componentes. Este proceso se ejecuta al dar click al botón Precatch Data que genera un esquema de base de datos con el nombre de DBLOD, el cual contiene las tablas como se muestran en la figura 4.24.

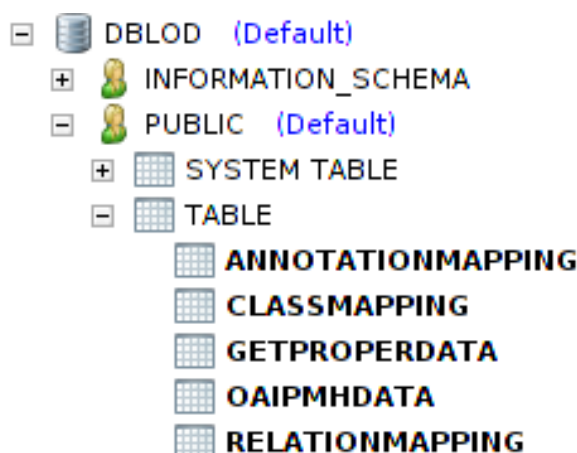


Figura 4.24: Esquema DBLOD en la base de datos H2

Las tablas creadas en el esquema DBLOD para los componentes **OAILoader** y **GET Properties OWL** contienen una clave primaria conformada por tres campos, los cuales se detallan a continuación.

- **TRANSID**: Esta columna almacena el nombre de la transformación creada en el PDI.
- **STEPID**: Esta columna almacena el nombre del STEP (Componente).
- **SEQUENCE**: Esta columna almacena el valor de una secuencia creada para cada registro.

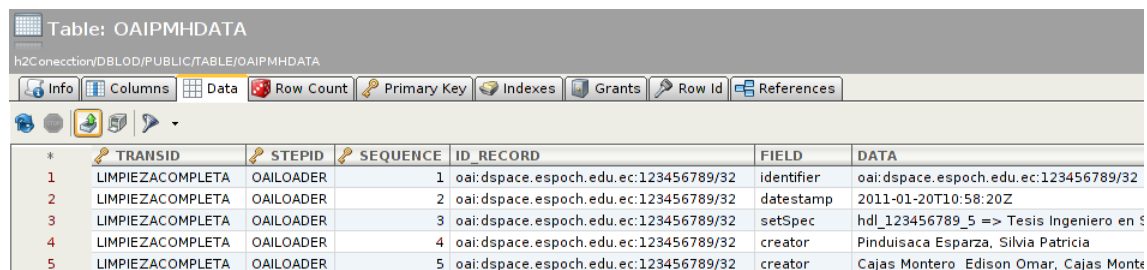
4.9.1.1. Tabla OAIPMHDATA

La finalidad de esta tabla es almacenar la registros obtenidos del componente OAIPMH. Para ser utilizado en el componente de mapeo y de generación de RDF:

Los campos de mayor relevancia para esta tabla se detallan a continuación.

- **Idrecord**: Esta columna almacena el identificador de cada registro extraído del repositorio.
- **Field**: Esta columna almacena el nombre del campo que se representa en el registro.
- **Data**: Esta columna almacena el valor en si del registro que se esta recuperando.

Finalmente en la figura 4.25 se muestra la manera de como se almacenan los registros en la tabla, los cuales son creados mediante el botón de precargar del componente de OAIPMH.



#	TRANSID	STEPSID	SEQUENCE	ID_RECORD	FIELD	DATA
1	LIMPIEZACOMPLETA	OAILOADER	1	oai:dspace.espace.edu.ec:123456789/32	identifier	oai:dspace.espace.edu.ec:123456789/32
2	LIMPIEZACOMPLETA	OAILOADER	2	oai:dspace.espace.edu.ec:123456789/32	datestamp	2011-01-20T10:58:20Z
3	LIMPIEZACOMPLETA	OAILOADER	3	oai:dspace.espace.edu.ec:123456789/32	setSpec	hdl_123456789_5 => Tesis Ingeniero en S
4	LIMPIEZACOMPLETA	OAILOADER	4	oai:dspace.espace.edu.ec:123456789/32	creator	Pinduisaca Esparza, Silvia Patricia
5	LIMPIEZACOMPLETA	OAILOADER	5	oai:dspace.espace.edu.ec:123456789/32	creator	Cajas Montero Edison Omar, Cajas Mont

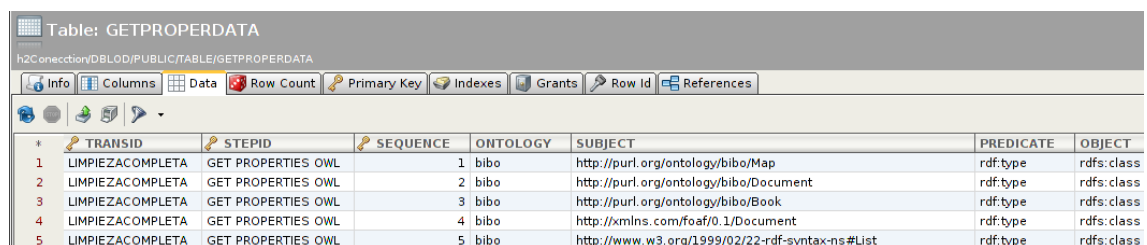
Figura 4.25: Tabla OAIPMHDATA del esquema DBLOD en H2

4.9.1.2. Tabla GETPROPERDATA

La finalidad de esta tabla es almacenar la registros obtenidos del componente **GET Properties OWL**, para ser utilizado en el componente de mapeo y de generación de RDF. Los campos de mayor relevancia para esta tabla se detallan a continuación.

- **Ontology:** Esta columna almacena el nombre de la ontología a la que pertenece el registro.
- **Subject:** Esta columna almacena la URL de la clase o propiedad listada.
- **Object:** Esta columna almacena los `rdfs:class` o `rdfs:properties` dependiendo si el registro es una clase o propiedad. Por lo tanto la columna permite distinguir si se trata de una clase o propiedad.

Finalmente cmo se muestra en la figura 4.26 la manera de como se almacenan los registros en la tabla, los cuales son creados mediante el botón de precargar del componente de **GET Properties OWL**.



#	TRANSID	STEPSID	SEQUENCE	ONTOLOGY	SUBJECT	PREDICATE	OBJECT
1	LIMPIEZACOMPLETA	GET PROPERTIES OWL	1	bibo	http://purl.org/ontology/bibo/Map	rdfs:type	rdfs:class
2	LIMPIEZACOMPLETA	GET PROPERTIES OWL	2	bibo	http://purl.org/ontology/bibo/Document	rdfs:type	rdfs:class
3	LIMPIEZACOMPLETA	GET PROPERTIES OWL	3	bibo	http://purl.org/ontology/bibo/Book	rdfs:type	rdfs:class
4	LIMPIEZACOMPLETA	GET PROPERTIES OWL	4	bibo	http://xmlns.com/foaf/0.1/Document	rdfs:type	rdfs:class
5	LIMPIEZACOMPLETA	GET PROPERTIES OWL	5	bibo	http://www.w3.org/1999/02/22-rdf-syntax-ns#List	rdfs:type	rdfs:class

Figura 4.26: Tabla GETPROPERDATA del esquema DBLOD en H2

Capítulo 5

Ejemplo práctico



UNIVERSIDAD DE CUENCA
desde 1867

5.1. Introducción

El objetivo principal de este capítulo es realizar un ejemplo práctico de uso del Framework de *Linked Open Data (LOD)* desarrollado en esta tesis. A continuación se muestra en la figura [5.1] la configuración de los componentes desarrollados para cada etapa.

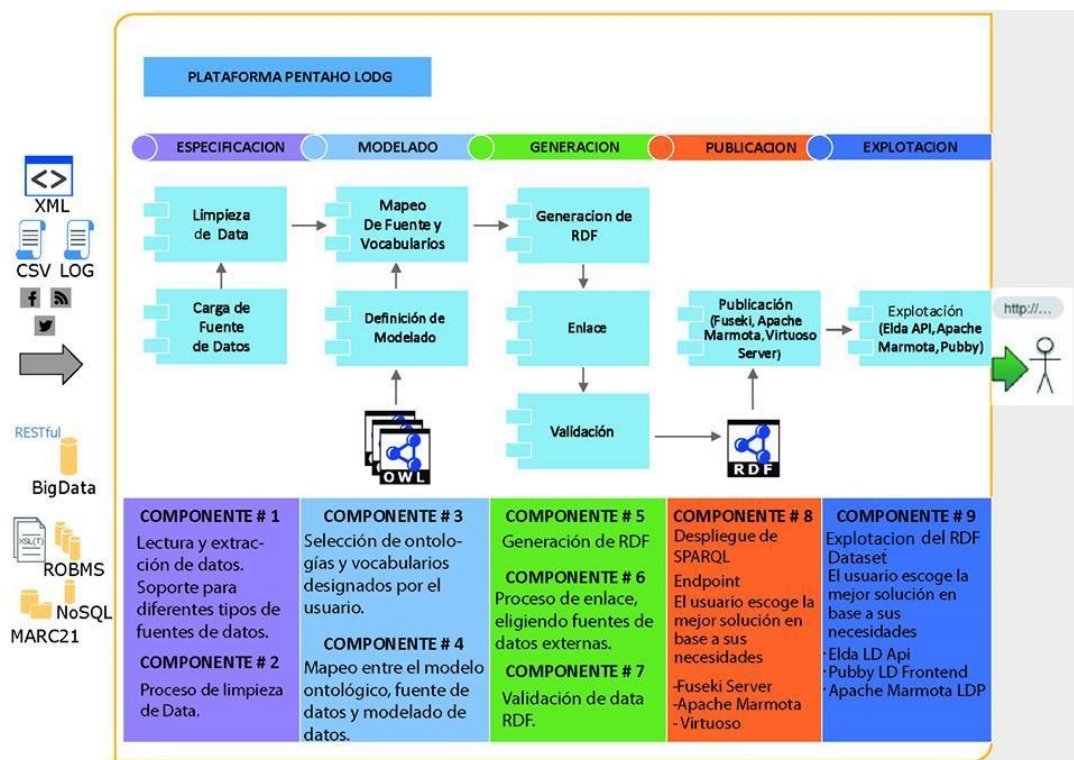


Figura 5.1: Figura *Framework para la generación de (LOD)* [3]

El caso práctico implementa la metodología *Linked Open Data (LOD)* [2.3].

5.2. Especificación

En la etapa de especificación se configura dos componentes, como se muestra en la figura [5.1]. El componente 1 permite la conexión y extracción de los datos, y el componente 2 que realiza la limpieza de los datos. Hay que decir, que el componente 1 en el proyecto toma el nombre de **OAILoader**, y el componente 2 son configuraciones entre componentes propios de PDI.

5.2.1. Configuración del Componente *OAILoader*

En esta sección se muestra la configuración de cada uno de los campos del componente *OAILoader* que le permite conectarse y extraer los datos desde los repositorios OAIPMH. En la figura [5.2] se muestra la configuración para este ejemplo.

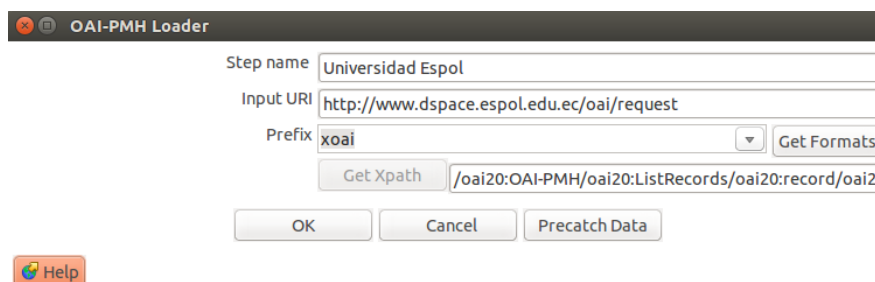


Figura 5.2: Configuración componente *OAILoader*

5.2.1.1. Parámetros de Entrada

Entre los parámetros que se deben configurar para el uso del componente *OAILoader* se tiene:

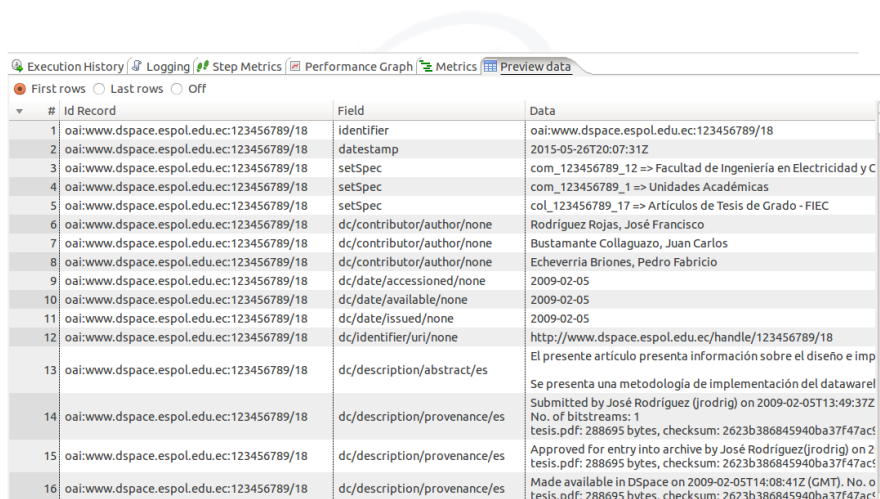
- StepName: Es el nombre que se asigna al componente en la transformación y en este caso será Universidad ESPOL.
- Input URI: Se especifican la URL del repositorio digital OAI para conectar y extraer los datos. Por ejemplo *http://www.dspace.espol.edu.ec/oai/request*, esta es la URL de la universidad ESPOL.
- Prefix: Esta opción especifica el formato para la extracción de Datos, para el ejemplo se toma xoai.
- Get Xpath: En este campo se asigna al componente una XPath para extraer todos los registros o solamente un campo específico. En este ejemplo se configura el XPath */oai20:OAI-PMH/oai20:ListRecords/oai20:record/oai20:metadata/xoai:metadata/xoai:element[@name='dc']* que permite la extracción de todos los registros con todos sus atributos.

Nota: el botón Precatch Data permite almacenar los datos extraídos del repositorio en la base embebida descrita en la sección 4.9. Esta base de datos se usa en los componentes de modelado y generación para realizar el mapeo de los datos y la generación en RDF.

5.2.1.2. Salida del Componente

El resultado de este componente es una tabla con 3 columnas que permite visualizar los datos extraídos del repositorio, como se muestra en la figura [5.3]. En donde:

- Idrecord: Es el Id del registro en repositorio, en el caso actual serán los creados dentro del repositorio de la ESPOL.
- Field: Especifica el nombre del campo que se representa en el registro.
- Data: Es el valor del registro que se está recuperando.



#	Id Record	Field	Data
1	oai:www.dspace.espol.edu.ec:123456789/18	identifier	oai:www.dspace.espol.edu.ec:123456789/18
2	oai:www.dspace.espol.edu.ec:123456789/18	timestamp	2015-05-26T20:07:31Z
3	oai:www.dspace.espol.edu.ec:123456789/18	setSpec	com_123456789_12 => Facultad de Ingeniería en Electricidad y C
4	oai:www.dspace.espol.edu.ec:123456789/18	setSpec	com_123456789_1 => Unidades Académicas
5	oai:www.dspace.espol.edu.ec:123456789/18	setSpec	col_123456789_17 => Artículos de Tesis de Grado - FIEC
6	oai:www.dspace.espol.edu.ec:123456789/18	dc/contributor/author/none	Rodríguez Rojas, José Francisco
7	oai:www.dspace.espol.edu.ec:123456789/18	dc/contributor/author/none	Bustamante Collaguazo, Juan Carlos
8	oai:www.dspace.espol.edu.ec:123456789/18	dc/contributor/author/none	Echeverría Briones, Pedro Fabricio
9	oai:www.dspace.espol.edu.ec:123456789/18	dc/date/accessioned/none	2009-02-05
10	oai:www.dspace.espol.edu.ec:123456789/18	dc/date/available/none	2009-02-05
11	oai:www.dspace.espol.edu.ec:123456789/18	dc/date/issued/none	2009-02-05
12	oai:www.dspace.espol.edu.ec:123456789/18	dc/identifier/uri/none	http://www.dspace.espol.edu.ec/handle/123456789/18
13	oai:www.dspace.espol.edu.ec:123456789/18	dc/description/abstract/es	El presente artículo presenta información sobre el diseño e imp
14	oai:www.dspace.espol.edu.ec:123456789/18	dc/description/provenance/es	Se presenta una metodología de implementación del datawarel Submitted by José Rodríguez (jrodrig) on 2009-02-05T13:49:37Z No. of bitstreams: 1 tesis.pdf: 288695 bytes, checksum: 2623b386845940ba37f47ac
15	oai:www.dspace.espol.edu.ec:123456789/18	dc/description/provenance/es	Approved for entry into archive by José Rodríguez(jrodrig) on 2 tesis.pdf: 288695 bytes, checksum: 2623b386845940ba37f47ac
16	oai:www.dspace.espol.edu.ec:123456789/18	dc/description/provenance/es	Made available in DSpace on 2009-02-05T14:08:41Z (GMT). No. o tesis.pdf: 288695 bytes, checksum: 2623b386845940ba37f47ac

Figura 5.3: Salida del componente OAIloader

Como se muestra en la figura [5.1], el siguiente paso es la configuración del componente 2, el cual toma la salida del componente **OAIloader** como su entrada. Hay que recalcar, que inicialmente el componente 2 se encontraba en la etapa de generación a continuación del componente **R2RMLtoRDF**, pero se determinó llevarlo a la etapa de especificación porque es más factible limpiar los datos antes de generar el RDF.



Figura 5.4: Transformación ejemplo práctico, componente **OAIloader**

5.2.2. Limpieza de Data

De acuerdo al Framework [5.1] en esta sección se realiza el proceso de *Limpieza de Data*. Este no es un componente desarrollado como parte del Framework, sino más bien son configuraciones de componentes de PDI, los cuales permiten corregir errores en los datos extraídos del repositorio, que para este ejemplo corresponde al de la universidad ESPOL. Por lo tanto, esta sección muestra los bloques de componentes configurados para realizar la limpieza de los datos y que tipo de errores fueron solucionados.

5.2.2.1. Bloques de Componentes para la Limpieza en PDI

Para comenzar la limpieza de los datos, en el ejemplo se tiene una estructura que deben cumplir para ser considerados como correctos, en este ejemplo la estructura de los nombres de autores debe ser Apellido1 Apellido2, Nombre1 Nombre2, por ejemplo, un nombre correcto es Santacruz Mora, Edison Freddy. Adicionalmente, se debe verificar que los nombres no contengan caracteres referentes a acentos especiales, diéresis, tildes o algún error de digitación. Ahora bien, en esta sección se muestra como varios componentes de PDI configurados en bloques solucionan los errores encontrados en este ejemplo particular.

Selección de Datos con Estructura Correcta

En este bloque se verifica que los datos estén con la estructura correcta como se especifico anteriormente, además se corrigen varios errores de digitación que se han encontrado. La figura 5.5 muestra el bloque para este ejemplo, los datos que no cumplan este primer filtro son la entrada del siguiente bloque y los datos que cumplan son la entrada al componte *Data Precaching* detallado en sección 5.3.1.



Figura 5.5: Bloque para comprobar la estructura en los datos.

A continuación se detallan los componentes de PDI utilizados en este bloque:

- Replace in string: Es un componente que permite buscar un string para remplazarlo con otro string o quitarlo definitivamente. Errores corregidos con este componente:

1. Error de nombres con 2 comas como separador de la estructura.
2. Error de nombres con acentos especiales, diéresis y tildes.
3. Error de nombres con guiones bajos o medios.

En la figura 5.6 se muestra la configuración de este componente solucionando uno de los errores anteriormente detallados.

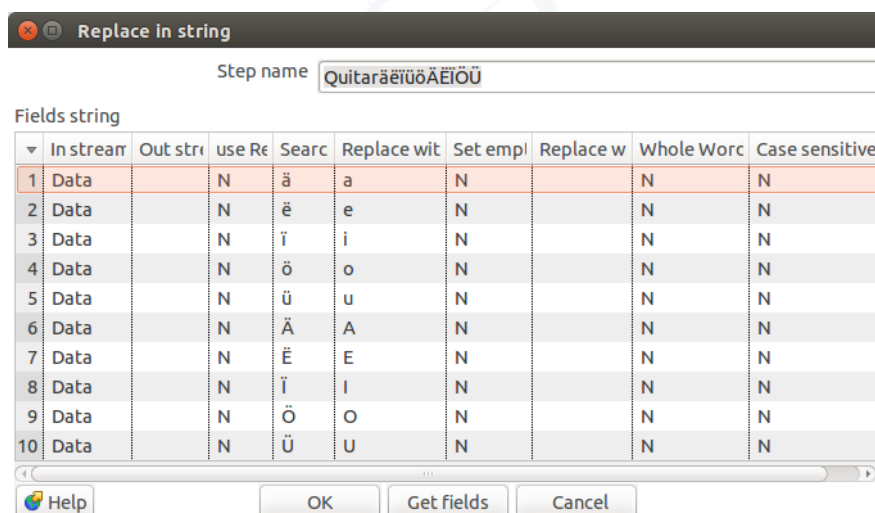


Figura 5.6: Configuración del componente Replace in string.

- Regex Evaluation 2: Este componente permite evaluar una expresión regular, que para este bloque evalúa si la estructura de los nombres de autores es la correcta. La figura 5.7 muestra una expresión regular para evaluar la estructura de los nombres, esta es configurada en el componente.

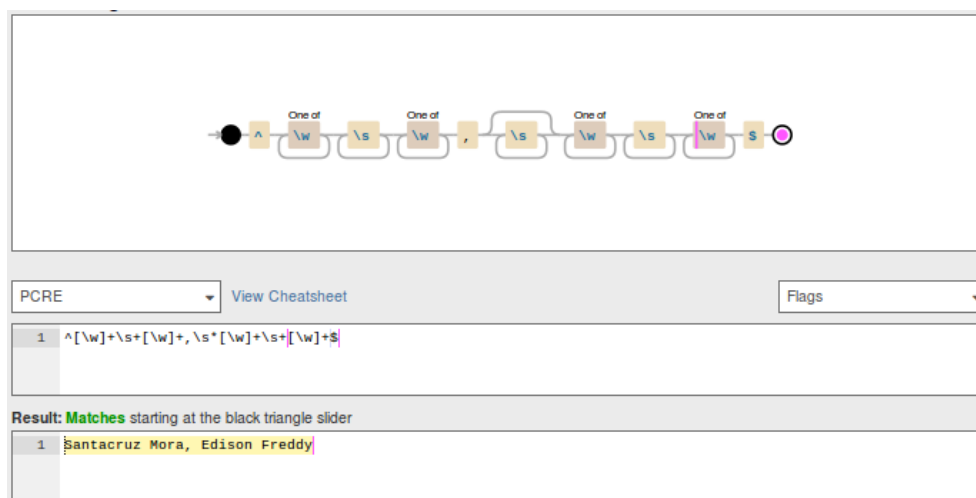


Figura 5.7: Ejemplo de una expresión regular evaluando la estructura de los nombres.

Comprobación de Datos con Estructura Correcta

En este bloque se corrigen los nombres de autores que no cumplieron la estructura descrita en la sección 5.2.2.1, pero sin embargo, son nombres validos con otras estructuras, tales como: Apellido1 Apellido2, Nombre1 ó Apellido1, Nombre1, los cuales se filtran en este bloque. La figura 5.9 muestra el bloque para este ejemplo, los datos que no cumplan este nuevo filtro pasan a una lista para ser verificados manualmente y los datos que cumplan se agregar al componte *Data Precaching* detallado en sección 5.3.1.

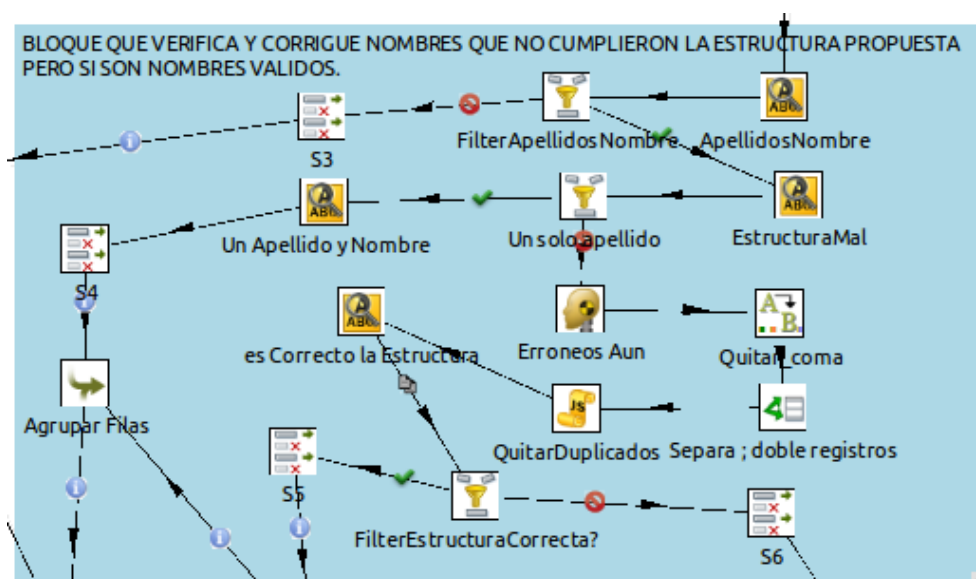


Figura 5.8: Bloque para evaluar otras estructuras validas en los datos.

Este bloque utilizo los siguientes componentes:

- **Regex Evaluation:** Este componente utiliza diferentes expresiones regulares para evaluar las nuevas estructuras validas.
- **Modified Java Script Value:** Este componente permite programar en el lenguaje JavaScript. El error que se corrige con este componente es el de apellidos o nombres duplicados por ejemplo: Santacruz Santacruz Mora Mora, Edison Edison Freddy Freddy, como salida se obtiene los nombres con las estructuras antes mencionadas de acuerdo al caso. La figura 5.9 muestra la configuración descrita para este componente.



Figura 5.9: Configuración del componente Modified Java Script Value para corregir duplicados.

Nota: toda la configuración descrita en esta sección se representa con un recuadro azul con el nombre de Limpieza de datos, en las siguientes secciones de este ejemplo practico.

5.3. Componentes Utilitarios

Los componentes descritos en esta sección no fueron desarrollados en esta tesis, pero se deben configurar en la transformación porque son necesario para la ejecución de este ejemplo, estos proveen diferentes funcionalidades que se detallan en esta sección.

5.3.1. Configuración Componente *Data Precatching*

Esta sección detalla la configuración del componente *Data Precatching*, el cual se encarga de guardar los datos de salida del proceso *limpieza de Data* descrito en la sección 5.2.2. El componente almacena los datos en la tabla OAIPMHDATA de la base embebida descrita en la sección 4.9.1.1. La información que se almacena, se utiliza en los componentes de *Ontology Mapping* y *R2RMLtoRDF*. La configuración para este ejemplo se muestra en la figura 5.10.

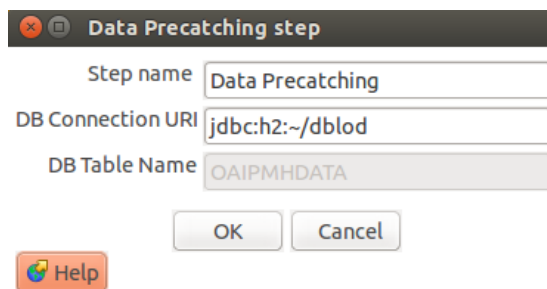


Figura 5.10: Configuración componente *Data Precatching*.

5.3.1.1. Parámetros del Componente

- StepName: Es el nombre que se asignará al componente para la transformación, para el ejemplo es *Data Precatching*.
- DB Connection URI : Es la URI de conexión con la base de datos jdbc:h2: /dblod. Este valor se carga automáticamente y no es editable.
- DB Table Name : Es el nombre de la tabla dentro de la base embebida, este valor se configura automáticamente y no es editable.

5.3.2. Uso del Componente *Block this step until steps finish*

Este componente es parte de PDI, el cual detiene la ejecución de los componentes configurados que están definidos antes. La razón del uso de este componente es permitir cargar los datos y las ontologías antes de continuar las demás etapas del proceso de *Linked Data*. En la figura 5.11 se muestra la configuración descrita para este ejemplo.

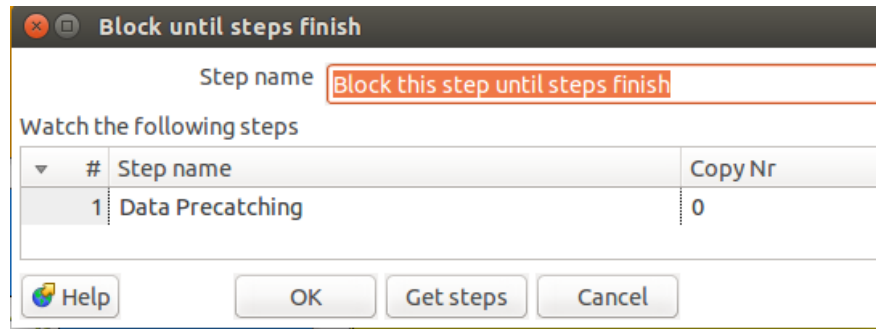


Figura 5.11: Configuración componente Block this step until steps finish

5.3.2.1. Parámetros de Entrada

- StepName: Es el nombre que se asignará al componente: Block this step until steps finish
- Columna StepName: Es el componente que debe terminar su ejecución, para continuar el proceso de la transformación, para este ejemplo *Data Precatching*.

5.3.2.2. Salida del Componente

El resultado de este componente es indicar a la transformación que debe esperar que finalice el componente *Data Precatching*, para continuar con la ejecución del resto de componentes.

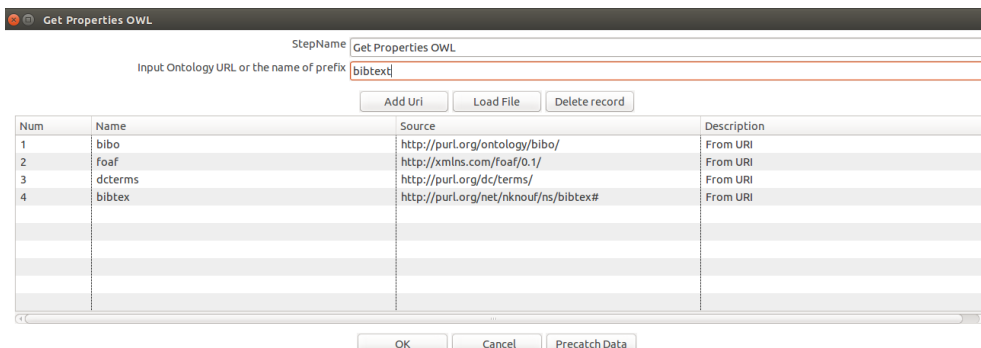
5.4. Modelado

En esta etapa se determina las ontologías adecuadas a ser utilizadas para relacionar los datos obtenidos en la etapa de especificación. En base a la metodología [5.1], se configurarán el componente 3 y el componente 4 para realizar la etapa de modelado .

5.4.1. Configuración del Componente para la Carga de las Ontologías

En esta sección se detalla un ejemplo práctico de la configuración y uso del componente de carga de ontologías. De acuerdo al Framework [5.1] aquí se con-

figurará el componente 3 para seleccionar ontologías. La figura [5.12] muestra la configuración para este ejemplo.



Num	Name	Source	Description
1	bibo	http://purl.org/ontology/bibo/	From URI
2	foaf	http://xmlns.com/foaf/0.1/	From URI
3	dcterms	http://purl.org/dc/terms/	From URI
4	bibtex	http://purl.org/net/nknouf/ns/bibtex#	From URI

Figura 5.12: Configuración componente *Get Properties OWL*

5.4.1.1. Parámetros de Entrada

Entre los parámetros que se deben configurar para el uso del componente *Get Properties OWL* se tiene :

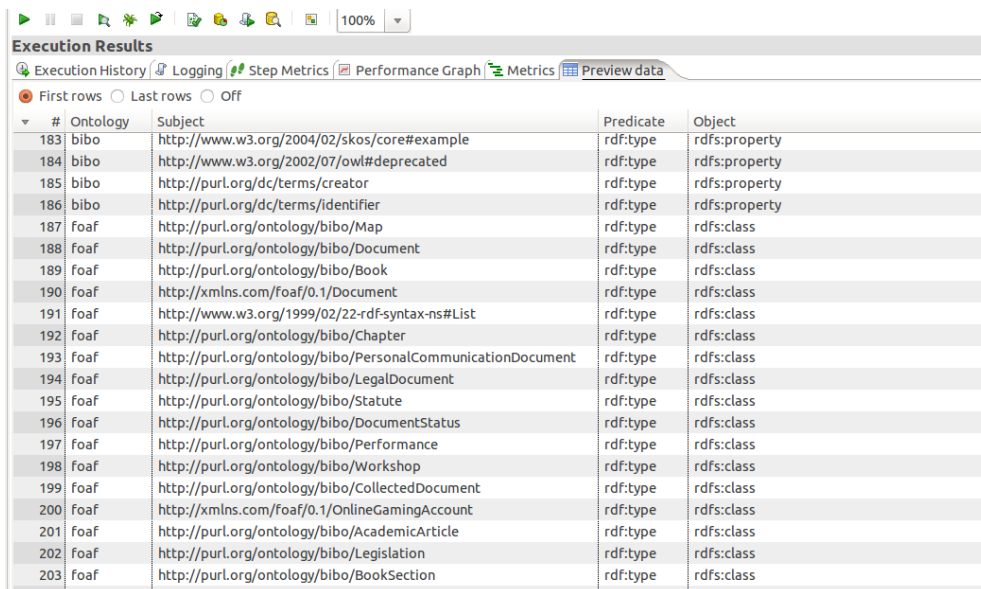
- StepName: Es el nombre que se asigna al componente para la transformación y en este caso será *Get Properties OWL*.
- Input Ontology URL or the Name or of Prefix: Se especifican los nombre de las ontologías en esta caso son: bibo, foaf, dcterms, bibtex. Cada ontología se cargo al accionar el botón Add URI.
- Boton Load File : Esta opción se utiliza para cargar ontologías desde archivos OWL, RDF o TLL, para el ejemplo actual no fue necesario porque las ontologías se cargaron con su URI.

5.4.1.2. Salida del Componente

El resultado al ejecutar el componente es un listado con las clases y propiedades de cada ontología ingresada, este resultado esta cargado en 4 columnas, que se detallan a continuación.

- Ontology: Es el nombre de la ontología a la que corresponde la propiedad o fila, en nuestro caso será bibo, foaf, dcterms, bibtex.
- Subject : Es la URI de la propiedad o clase listada.

- Predicate : Identifica el tipo.
- Object : Puede tener dos valores `rdfs:class` o `rdfs:property`.



#	Ontology	Subject	Predicate	Object
183	bibo	http://www.w3.org/2004/02/skos/core#example	rdf:type	rdfs:property
184	bibo	http://www.w3.org/2002/07/owl#deprecated	rdf:type	rdfs:property
185	bibo	http://purl.org/dc/terms/creator	rdf:type	rdfs:property
186	bibo	http://purl.org/dc/terms/identifier	rdf:type	rdfs:property
187	foaf	http://purl.org/ontology/bibo/Map	rdf:type	rdfs:class
188	foaf	http://purl.org/ontology/bibo/Document	rdf:type	rdfs:class
189	foaf	http://purl.org/ontology/bibo/Book	rdf:type	rdfs:class
190	foaf	http://xmlns.com/foaf/0.1/Document	rdf:type	rdfs:class
191	foaf	http://www.w3.org/1999/02/22-rdf-syntax-ns#List	rdf:type	rdfs:class
192	foaf	http://purl.org/ontology/bibo/Chapter	rdf:type	rdfs:class
193	foaf	http://purl.org/ontology/bibo/PersonalCommunicationDocument	rdf:type	rdfs:class
194	foaf	http://purl.org/ontology/bibo/LegalDocument	rdf:type	rdfs:class
195	foaf	http://purl.org/ontology/bibo/Statute	rdf:type	rdfs:class
196	foaf	http://purl.org/ontology/bibo/DocumentStatus	rdf:type	rdfs:class
197	foaf	http://purl.org/ontology/bibo/Performance	rdf:type	rdfs:class
198	foaf	http://purl.org/ontology/bibo/Workshop	rdf:type	rdfs:class
199	foaf	http://purl.org/ontology/bibo/CollectedDocument	rdf:type	rdfs:class
200	foaf	http://xmlns.com/foaf/0.1/OnlineGamingAccount	rdf:type	rdfs:class
201	foaf	http://purl.org/ontology/bibo/AcademicArticle	rdf:type	rdfs:class
202	foaf	http://purl.org/ontology/bibo/Legislation	rdf:type	rdfs:class
203	foaf	http://purl.org/ontology/bibo/BookSection	rdf:type	rdfs:class

Figura 5.13: Salida del componente *Get Properties OWL*

Finalmente se debe hacer clic en el Botón Precatch Data para que los demás componentes de la transformación tengan acceso a las ontologías seleccionadas.

El estado de la transformación del ejemplo práctico al termino de la configuración del componente *Get Properties OWL* se muestra en la figura [5.14].

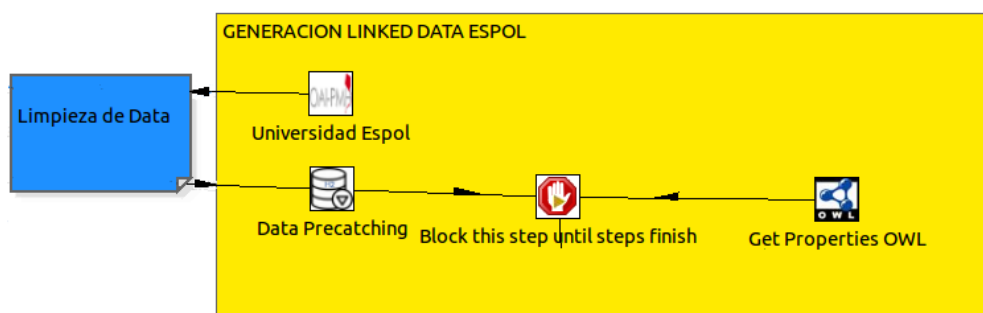
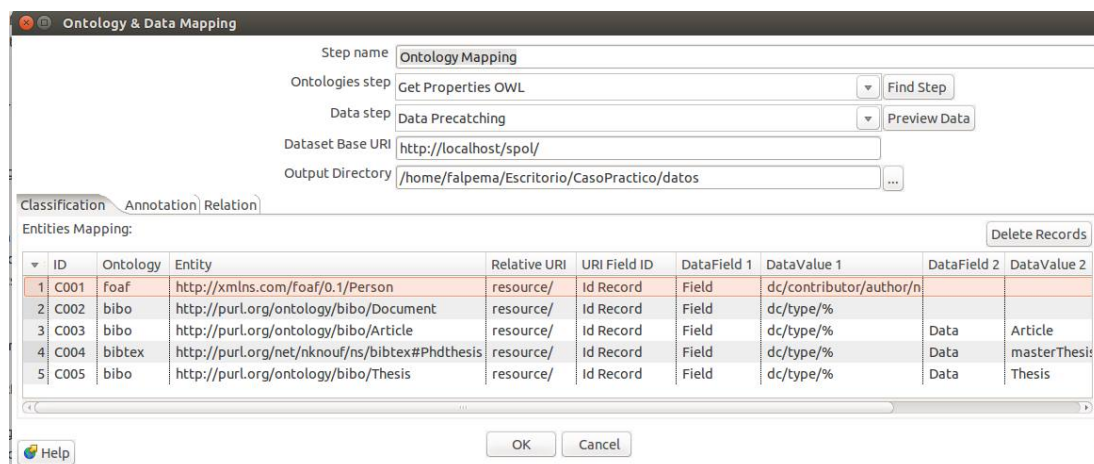


Figura 5.14: Transformación ejemplo práctico, componente *Get Properties OWL*.

5.4.2. Configuración del Componente *Ontology Mapping*

De acuerdo al Framework [5.1] en esta sección se configura el componente 4, el cual se encarga del mapeo entre los datos y ontologías. Por lo tanto este

recibe dos entradas: la primera es la fuente de Datos del repositorio de la ESPOL proporcionada por el componente 1 y la segunda son las ontologías configuradas en el componente 2. En la figura [5.15] se muestra la configuración para este ejemplo.



ID	Ontology	Entity	Relative URI	URI Field ID	DataField 1	DataValue 1	DataField 2	DataValue 2
1 C001	foaf	http://xmlns.com/foaf/0.1/Person	resource/	Id Record	Field	dc/contributor/author/n		
2 C002	bibo	http://purl.org/ontology/bibo/Document	resource/	Id Record	Field	dc/type/%		
3 C003	bibo	http://purl.org/ontology/bibo/Article	resource/	Id Record	Field	dc/type/%	Data	Article
4 C004	bibtex	http://purl.org/net/nknouf/ns/bibtex#Phdthesis	resource/	Id Record	Field	dc/type/%	Data	masterThesi
5 C005	bibo	http://purl.org/ontology/bibo/Thesis	resource/	Id Record	Field	dc/type/%	Data	Thesis

Figura 5.15: Configuración componente *Ontology Mapping*

5.4.2.1. Parámetros Base

Este componente necesita la configuración de varios parámetros previo a realizar el mapeo. La figura [5.16] muestra la configuración de los mismos.

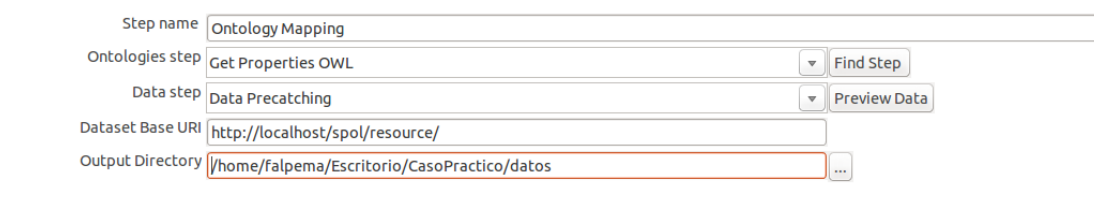


Figura 5.16: Configuración parámetros base del componente *Ontology Mapping*.

La configuración de los parámetros base se detalla a continuación:

- **StepName:** Es el nombre que se asigna al componente para la transformación y en este caso será *Ontology Mapping*.
- **Ontologies Step :** Se especifica el nombre del componente de donde se carga las ontologías, para el caso actual se selecciona a *Get Properties OWL*.

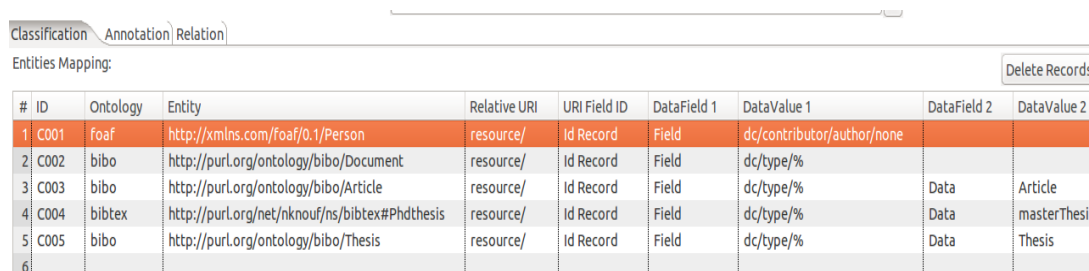
- Data step : Se especifica el nombre del componente de donde se obtiene la fuente de Datos, para el ejemplo actual es el componente *Data Precatching*.
- Dataset Base URI : Se especifica la base URI <http://localhost/esp/> para el mapeo actual.
- Output Directory : Se especifica el directorio en donde se genera el archivo de mapeo en lenguaje R2RML. para el ejemplo la salida es *Ontology Mapping-R2RML.ttl*.

Nota: La salida de ese componente es un archivo con el nombre *Ontology Mapping-R2RML.ttl*, en la ruta */home/falpema/Escritorio/Casopráctico/datos*, configurado para este ejemplo.

El componente tiene 3 pestañas donde se realiza la configuración para el mapeo, como son: *Classification*, *Annotation* y *Relation*. A continuación se muestra como configurar cada una de ellas.

5.4.2.2. Configuración de la Pestaña *Classification*

Se selecciona la primera pestaña llamada *Clasificación* para realizar el mapeo de las entidades seleccionadas en el componente 3. Es decir aquí se identifica los datos del repositorio que se deben asignar a cada entidad. En la figura [5.17] se muestra la configuración descrita en este ejemplo.



#	ID	Ontology	Entity	Relative URI	URI Field ID	DataField 1	DataValue 1	DataField 2	DataValue 2
1	C001	foaf	http://xmlns.com/foaf/0.1/Person	resource/	Id Record	Field	dc/contributor/author/none		
2	C002	bibo	http://purl.org/ontology/bibo/Document	resource/	Id Record	Field	dc/type/%		
3	C003	bibo	http://purl.org/ontology/bibo/Article	resource/	Id Record	Field	dc/type/%	Data	Article
4	C004	bibtex	http://purl.org/net/nknouf/ns/bibtex#Phdthesis	resource/	Id Record	Field	dc/type/%	Data	masterThesis
5	C005	bibo	http://purl.org/ontology/bibo/Thesis	resource/	Id Record	Field	dc/type/%	Data	Thesis
6									

Figura 5.17: Configuración Primera Pestaña *Clasificación*.

Como ejemplo se realiza el mapeo la entidad *Person* de la ontología *FOAF*. En este caso se configuran los siguiente campos:

- ID : Esta columna contiene automáticamente un código generado de identificación para la entidad mapeada, en esta caso es C001.

- **Ontology** : Se selecciona la Ontología que se desea mapear, en el caso del ejemplo es FOAF.
- **Entity** : De acuerdo a la ontología se selecciona la entidad con la que se mapean los recursos, para el ejemplo es `http://xmlns.com/foaf/0.1/Person`.
- **Relative URI** : Se especifica la URI relativa `resource/`, esta se adjunta a la descrita en el Dataset Base URI `http://localhost/esp/`.
- **URI Field ID** : Se selecciona el campo por el cual se identifica los recursos de la Fuente de Datos, en este caso se escoge `IdRecord` porque en el repositorio de la ESPOL este campo es el identificador único de los datos.
- **DataField1** : Se selecciona el campo de la fuente de datos en donde se buscarán las personas a mapear, en este caso el campo es `Field`.
- **DataValue1** : Se selecciona el valor que debe tener el campo `Field` para ser mapeado como persona, para el ejemplo es `dc/contributor/author/none`.

Existen otros campos como `DataField2` y `DataValue2` que se encuentra vacíos, porque para este ejemplo no fueron necesarias mas condiciones de búsqueda para mapear la entidad `Person` con sus registros respectivos.

Nota: En la figura [5.17] se muestran otros ejemplos de mapeo para las entidades: `Document`, `Article`, `PhdThesis` y `Thesis`. Así pues pueden existir entidades que necesiten más condiciones para identificar sus registros dentro del repositorio, para esto existen mas columnas de especificación como `DataField3` y `DataValue3`.

5.4.2.3. Configuración de la Pestaña *Anotation*

Esta configuración corresponde a la segunda pestaña del componente, en donde se realiza el mapeo de las propiedades correspondientes a las entidades definidas en el pestaña anterior [5.4.2.2], es decir para cada propiedad se indican las condiciones que identifican los registros que le pertenecen. Con respecto al ejemplo se mapean las propiedades: ***name***, ***LastName*** y ***firstName*** pertenecientes a la ontología FOAF. En la figura [5.18] se muestra la configuración descrita para este ejemplo.

Classification Annotation Relation

Properties Mapping: Delete Records

#	ID	Entity ClassID	Ontology	Property	ExtractionField	DataField	DataValue	Data Type	Language
1	A001	C001	foaf	http://xmlns.com/foaf/0.1/name	Data	Field	dc/contributor/author/none		
2	A002	C001	foaf	http://xmlns.com/foaf/0.1/lastName	Data	Field	apellido		
3	A003	C001	foaf	http://xmlns.com/foaf/0.1/firstName	Data	Field	nombre		
4	A004	C002	dcterms	http://purLorg/dc/terms/title	Data	Field	dc/title/en		
5	A005	C002	dcterms	http://purLorg/dc/terms/dateSubmitted	Data	Field	dc/date/accessioned/none		
6	A006	C002	dcterms	http://purLorg/dc/terms/available	Data	Field	dc/date/available/none		
7	A007	C002	dcterms	http://purLorg/dc/terms/issued	Data	Field	dc/date/issued/none		
8	A008	C002	bibo	http://purLorg/ontology/bibo/abstract	Data	Field	dc/description/abstract/%		
9	A009	C002	dcterms	http://purLorg/dc/terms/provenance	Data	Field	dc/description/provenance/%		
10	A010	C002	dcterms	http://purLorg/dc/terms/subject	Data	Field	dc/subject/%		
11	A011	C002	dcterms	http://purLorg/dc/terms/title	Data	Field	dc/title/%		
12	A012	C002	dcterms	http://purLorg/dc/terms/language	Data	Field	dc/language/iso/%		
13	A013	C002	bibo	http://purLorg/ontology/bibo/uri	Data	Field	dc/identifier/uri/%		
14									

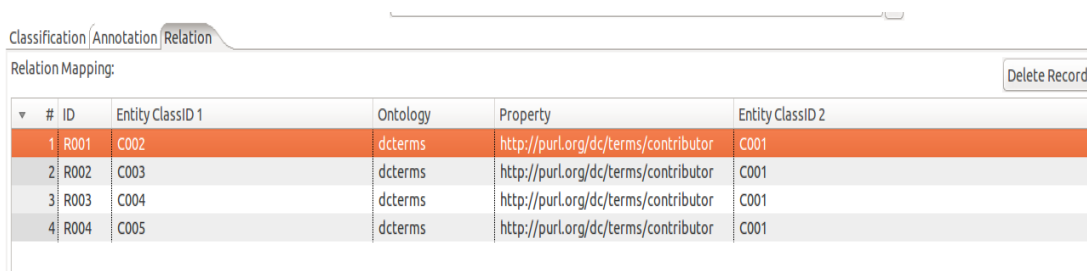
Figura 5.18: Configuración Segunda Pestaña de Anotación

Entre los parámetros que se deben configurar en el ejemplo son:

- ID: Esta columna genera automáticamente un código de identificación para la propiedad mapeada, en este caso es A001.
- Entity ClassID: Se selecciona el ClassId de la entidad a mapear, en este caso es C001 puesto que corresponde a la entidad Person.
- Ontology : Se selecciona la ontología que contiene la propiedad a mapear, en este caso es FOAF.
- Property : Se selecciona la propiedad Name por ser la propiedad a mapear.
- ExtractionField : Se selecciona el campo de donde se extrae el valor que se asigna a la propiedad, en este caso es Data que contiene los nombres de los Autores.
- DataField : Se selecciona el campo de la fuente de datos en donde se buscan las personas a mapear, en este caso es el campo Field.
- DataValue : Se selecciona el valor que debe tener el campo Field para ser mapeado como persona, en el ejemplo es dc/contributor/author/none para los datos de la ESPOL.
- DataType : Se selecciona el tipo de dato, en este caso es String.
- Language : Se selecciona el lenguaje, en este caso ES correspondiente a español.

5.4.2.4. Configuración de la Pestaña *Relation*

Esta configuración corresponde a la tercera pestaña del componente, en donde se realiza el mapeo que relaciona las entidades entre si. En este ejemplo práctico se desea mapear las entidades: Persona, Documento, Article, Thesis y PhdThesis, en donde una persona puede ser un contribuidor para Documentos, Artículos o Tesis, por lo tanto, se usara la propiedad *Contributor* para relacionar estas entidades. La figura [5.25] muestra la configuración descrita para este ejemplo.



#	ID	Entity ClassID 1	Ontology	Property	Entity ClassID 2
1	R001	C002	dcterms	http://purl.org/dc/terms/contributor	C001
2	R002	C003	dcterms	http://purl.org/dc/terms/contributor	C001
3	R003	C004	dcterms	http://purl.org/dc/terms/contributor	C001
4	R004	C005	dcterms	http://purl.org/dc/terms/contributor	C001

Figura 5.19: Configuración Tercera Pestaña

Los parámetros a configurar en este mapeo se muestran a continuación para la relación de que documento tiene un Contributor que es Person.

- ID: Esta columna genera automáticamente un código de identificación de la relacion mapeada, en este caso es R001.
- Entity ClassID 1: Se selecciona la entidad que inicia la relación a mapear, que en este caso es C002, el cual corresponde a la entidad Document.
- Ontology: De acuerdo a la ontología anterior se selecciona la entidad <http://xmlns.com/foaf/0.1/Person>.
- Property: Se selecciona la ontología que contiene la propiedad con la que se mapea la relacion, en este caso es dcterms.
- Entity ClassID 2: Se selecciona la Entidad con la cual se relaciona, la entidad seleccionada en el Entity ClassID 1, que en este caso es C001 es una entidad de tipo Person.

5.4.2.5. Salida del Componente

El resultado al ejecutar el componente es el mapeo entre ontologías y datos que se configuraron anteriormente, el cual crea un archivo en el lenguaje de mapeo

r2rml. Finalmente en base a la metodología este archivo será la entrada para el componente en la etapa de generación.

El estado de la transformación del ejemplo práctico al termino de la configuración del componente *Ontology Mapping* se muestra en la figura [5.20].

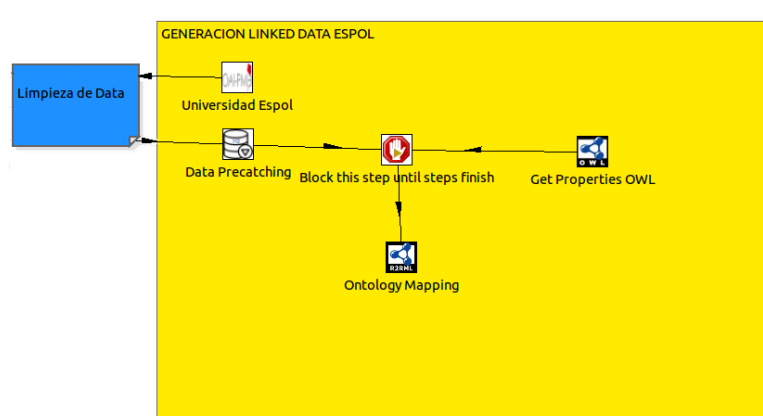


Figura 5.20: Estado de la transformación ya configurado el componente *Ontology Mapping*

5.5. Generación

En base a la metodología LOD [5.1], se configura el componente 5 que tiene el nombre de ***R2RMLtoRDF*** para realizar la generación de RDF, en este ejemplo práctico.

5.5.1. Configuración del componente ***R2RMLtoRDF***

En esta sección se configura el ***R2RMLtoRDF*** para la generación de los datos RDF, el cual recibe como entradas un archivo de mapeo en formato r2rml y una conexión a base de datos. En la figura 5.21 se muestra la configuración para este ejemplo.

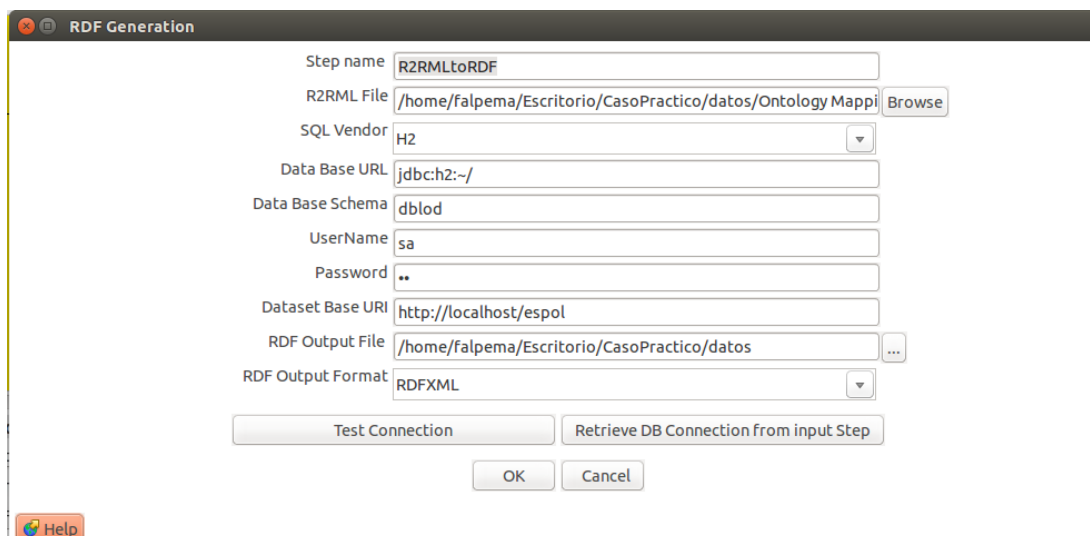


Figura 5.21: Configuración del Componente ***R2RMLtoRDF***

5.5.1.1. Configuración De Parámetros

A continuación se detalla los diferentes campos del componente ***R2RMLtoRDF*** que se deben configurar:

- StepName: nombre del step ***R2RMLtoRDF***, que además servirá como nombre al archivo de salida.
- R2RML File: directorio donde se encuentra ubicado el archivo de mapeo.
/home/falpema/Escritorio/Casopractico/datos/Ontology Mapping-R2RML.ttl
- SQL Vendor: driver de la base de datos que por lo general es H2.
- Data Base URL: URL de conexión a la base de datos que si es H2 es jdbc:h2: /
- Data Base Schema: nombre de la base de datos que si se utiliza H2 es dblod.
- UserName: nombre de usuario que si se conecta a la base embebida en los componentes es: sa
- Password: contraseña que si se conecta a la base embebida en los componentes es: sa
- Data Base URI: URI base para crear las tripletas RDF.
http://localhost/esp/

- RDF Output File: directorio para el archivo de salida con las tripletas RDF.
/home/falpema/Escritorio/Casopractico/datos.
- RDF Output Format: RDXML

5.5.1.2. Salida del Componente *R2RMLtoRDF*

Al ejecutar el componente se obtiene el archivo R2RMLtoRDF.rdf dentro del directorio /home/falpema/Escritorio/Casopractico/datos/. Finalmente en base a la metodología [5.1] este archivo RDF será la entrada para el componente en la etapa de publicación.

El estado de la transformación del ejemplo práctico al termino de la configuración del componente *R2RMLtoRDF* se muestra en la figura [5.22].

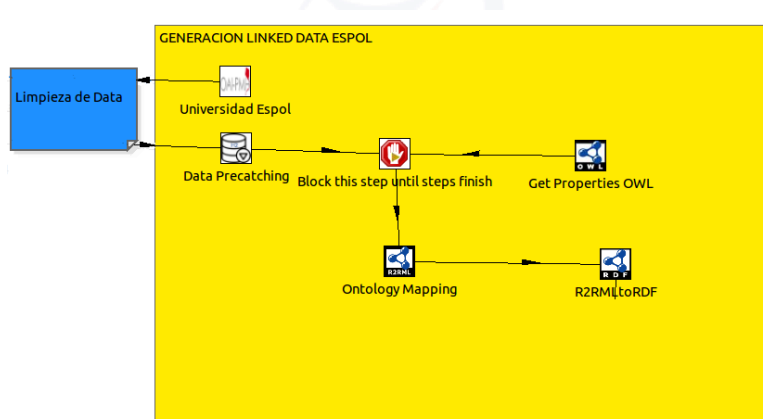


Figura 5.22: Transformación ejemplo práctico, componente R2RMLtoRDF

5.6. Publicación

En base a la metodología [5.1] se configura el componente 8 el cual tiene por nombre *Fuseki Loader*, para realizar la etapa de publicación para este ejemplo práctico.

5.6.1. Configuración del Componente *Fuseki Loader*

De acuerdo al Framework [5.1] aquí se configura el componente *Fuseki Loader*, el cual permite generar una instancia de Fuseki, con lo cual se realiza la publicación del RDF en el triplestore de Fuseki. Además, este recibe como entrada un archivo RDF, el cual es generado en el componente *R2RMLtoRDF* [5.5]. La figura [5.23] muestra la configuración de este componente para el ejemplo.

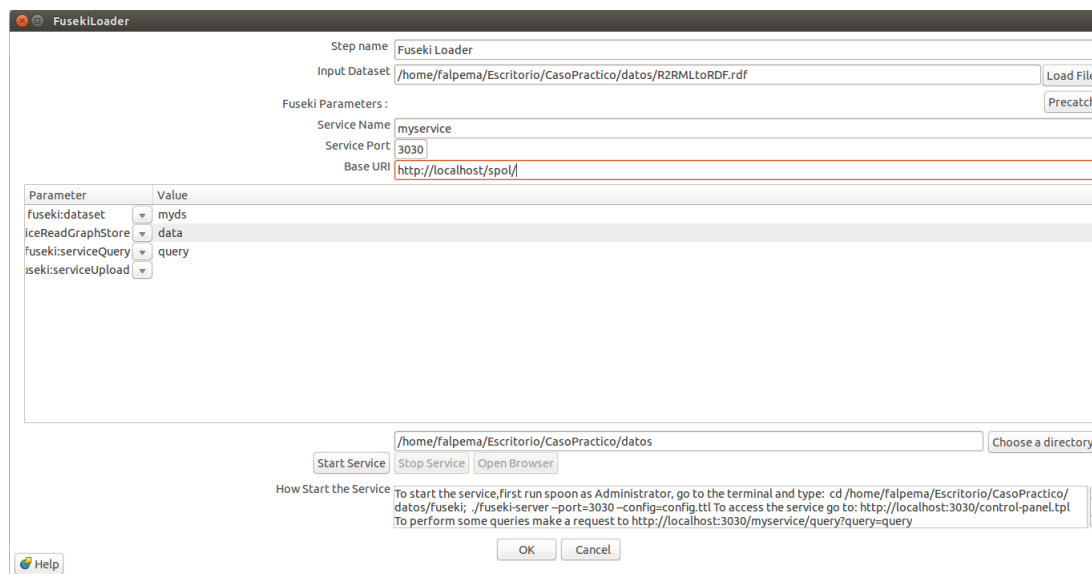


Figura 5.23: Configuración componente **Fuseki Loader**.

5.6.1.1. Configuración de Parámetros

Para este ejemplo práctico se realiza una configuración que levante el servicio localmente a través del puerto 3030 y permita el acceso al SPARQL EndPoint.

Los parámetros que se configuran en el componente son los siguientes.

- StepName: Es el nombre que se asigna al componente para la transformación, en este caso será **Fuseki Loader**.
- Input Dataset: Se selecciona el archivo que será el Dataset, en esta caso es /home/falpema/Escritorio/Casopractico/datos/R2RMLtoRDF.rdf.
Nota: por medio del botón Precatch se carga automáticamente este valor del componente anterior **R2RMLtoRDF**.
- Service Port: Es el puerto por donde se inicia el servicio de Fuseki, para el caso actual es 3030.
- Base URI: Es el grafo que se asigna a la información ingresada, para el caso actual seria http://localhost/spot/.
Nota: por medio del botón Precatch se carga automáticamente este valor del componente anterior.
- fuseki:dataset: Es el nombre que identificara el Dataset, para el caso actual se elige myds.

- `fuseki:serviceReadGraphStore`: Se refiere al protocolo para SPARQL Graph Store, para el caso actual seria data.
- `fuseki:serviceQuery`: Con esta propiedad se habilita el acceso al SPARQL Endpoint, por lo tanto para el presente caso se debe ingresar query.
- `Choose Directory`: Se selecciona el directorio para la instancia de Fuseki, en esta caso es `/home/falpema/Escritorio/Casopráctico/datos/`.

Nota: el componente permite parametrizar mas valores para Fuseki, los cuales para este ejemplo no son necesarios, esto se pueden consultar con mas detalle en la documentación oficial [9].

5.6.1.2. Salida del Componente *Fuseki Loader*

El resultado al ejecutar el componente es una instancia del servidor Fuseki configurado automáticamente, el cual permite publicar el RDF generado anteriormente. Todos los archivos del servidor se generan en el directorio que se configuró. Desde la interfaz del componente se puede iniciar el servicio SPARQL Endpoint, el cual es la entrada para el componente de explotación de datos.

Para comprobar que el servidor Fuseki fue generado correctamente: primero se inicia mediante el botón *StarService* y se accede a este con el botón *Open Browser*. La figura [5.24] muestra un navegador accediendo el servidor Fuseki creado.

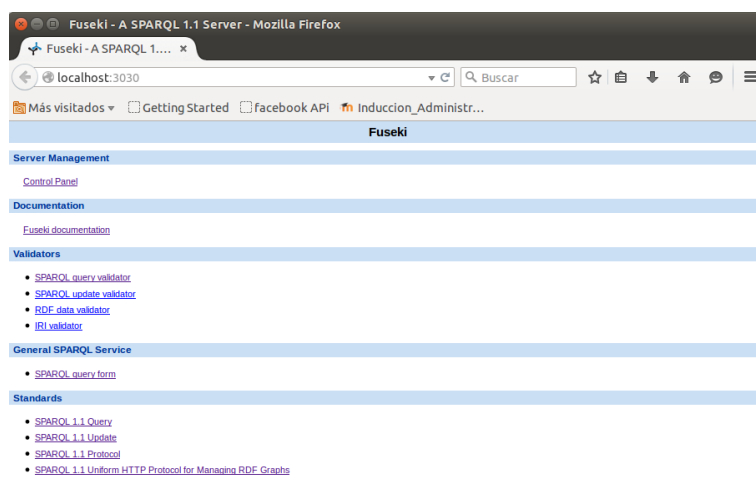


Figura 5.24: Navegador accediendo al servicio de Fuseki.

Ejecución del Servidor Fuseki

Una vez ejecutado el servidor Fuseki este permite realizar consultas SPARQL,

como ejemplo se ha realiza una consulta para obtener todas las entidades tipo objeto del Dataset consultado. La figura [5.25] muestra el servidor Fuseki y el resultado de la consulta de ejemplo.

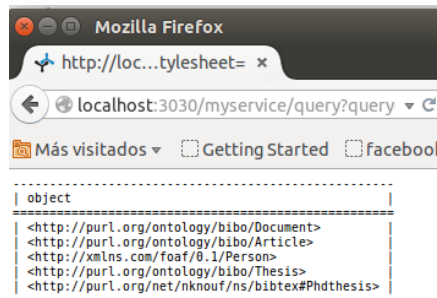


Figura 5.25: Resultado de la consulta SPARQL

El estado de la transformación del ejemplo práctico al termino de la configuración del componente **Fuseki Loader**, se muestra en la figura [5.26].

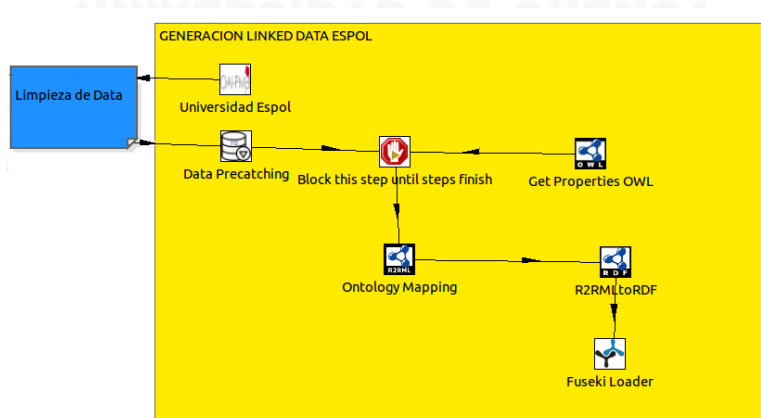


Figura 5.26: Transformación ejemplo práctico, componente **Fuseki Loader**

5.7. Explotación

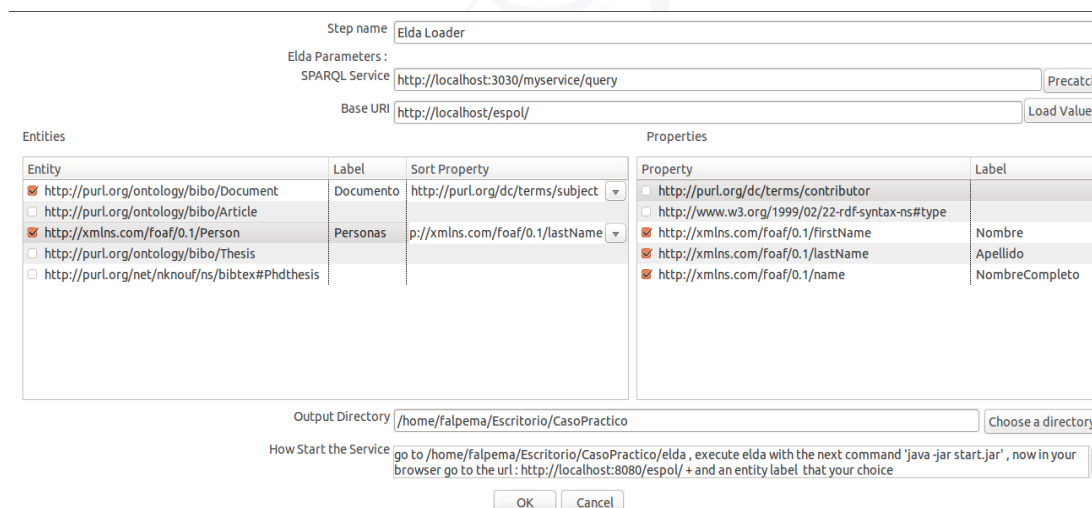
En base a la metodología [5.1], en esta sección se configura el componente 9 el cual tiene por nombre **Elda Step**, para realizar la etapa de Explotación de este ejemplo práctico.

5.7.1. Configuración del Componente *Elda Step*

De acuerdo al Framework [5.1] el componente recibe como entrada un servicio SPARQL EndPoint generado en el componente *Fuseki Loader* [5.7].

A través de este componente se genera una instancia de *Elda Step* para realizar la explotación del SPARQL Endpoint generado con el componente *Fuseki Loader*. Cabe recalcar que toda la configuración de esta instancia la genera automáticamente el componente en base a los parámetros ingresados.

Para este ejemplo practico se realiza una configuración de las entidades de tipo Person, la cuales muestran las propiedades: firstName, lastName y Name para cada recurso listado. La figura [5.27] muestra la configuración de este componente para el ejemplo.



Entity	Label	Sort Property
<input checked="" type="checkbox"/> http://purl.org/ontology/bibo/Document	Documento	http://purl.org/dc/terms/subject
<input type="checkbox"/> http://purl.org/ontology/bibo/Article		
<input checked="" type="checkbox"/> http://xmlns.com/foaf/0.1/Person	Personas	http://xmlns.com/foaf/0.1/lastName
<input type="checkbox"/> http://purl.org/ontology/bibo/Thesis		
<input type="checkbox"/> http://purl.org/net/nknouf/ns/bibtex#Phdthesis		

Property	Label
<input type="checkbox"/> http://purl.org/dc/terms/contributor	
<input type="checkbox"/> http://www.w3.org/1999/02/22-rdf-syntax-ns#type	
<input checked="" type="checkbox"/> http://xmlns.com/foaf/0.1/firstName	Nombre
<input checked="" type="checkbox"/> http://xmlns.com/foaf/0.1/lastName	Apellido
<input checked="" type="checkbox"/> http://xmlns.com/foaf/0.1/name	NombreCompleto

Figura 5.27: Configuración *Elda Step*

5.7.1.1. Configuración De Parámetros

Para este ejemplo práctico se realiza una configuración que levante el servicio localmente a través de puerto 8080.

Los parámetros que se configuran en el componente son los siguientes.

- StepName: Es el nombre que se asignará al componente para la transformación y en este caso será *Elda Step*.

- Service: Es la URL del SPARQL Endpoint a la accederá.

Nota: Por medio del botón Precatch se carga automáticamente este valor del componente *Fuseki Loader*.

- Directory: Es el directorio en donde se generara las instancia de Elda con todos sus archivo configurados, para el caso actual es /home/falpema/Escritorio/Casopráctico.

Al hacer click en el botón Load Values en la tabla Properties se cargaran todas las entidades del RDF, esto se realiza mediante una consulta SPARQL. Por cada fila se puede editar el nombre que se desea dar a la entidad dentro de la interfaz de Elda. En este ejemplo se agregaron los siguientes nombres : Documentos y Personas . Adicionalmente en la tabla Entity, se debe activar el check de las entidades que se desean mostrar, para el caso actual se activa para Person y Document. Para las entidad Person se activo las propiedades firstName, lastName y Name de la tabla Properties.

5.7.1.2. Salida del Componente *Elda Step*

El resultado al ejecutar el componente es una instancia de Elda configurada automáticamente , todos los archivos de esta instancia se generaran dentro del directorio que se escogió en los parámetros. Para el caso actual se puede iniciar Elda localmente con el fin de acceder a su entorno web, donde los usuarios podrán navegar con una interfaz amigable que muestra todos los recursos generados.

La figura [5.28] muestra un navegador accediendo al instancia de Elda generada en este ejemplo para los recursos de tipo Person.

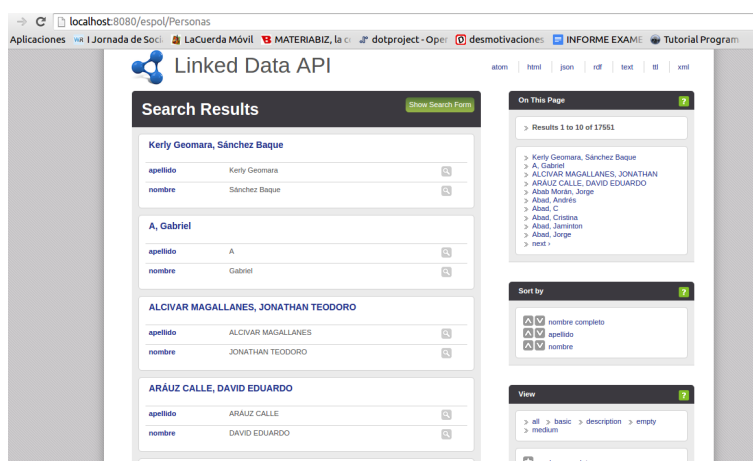


Figura 5.28: Servicio levantado en Elda configurado en este ejemplo para las entidades de tipo Person.

5.8. Configuración Finalizada

Todo los componentes se interconectan entre si en base a la metodología LOD [5.1], por lo tanto se han configurado en el orden que correspondían. Finalmente toda la configuración de este ejemplo práctico se muestra en la figura [5.29].

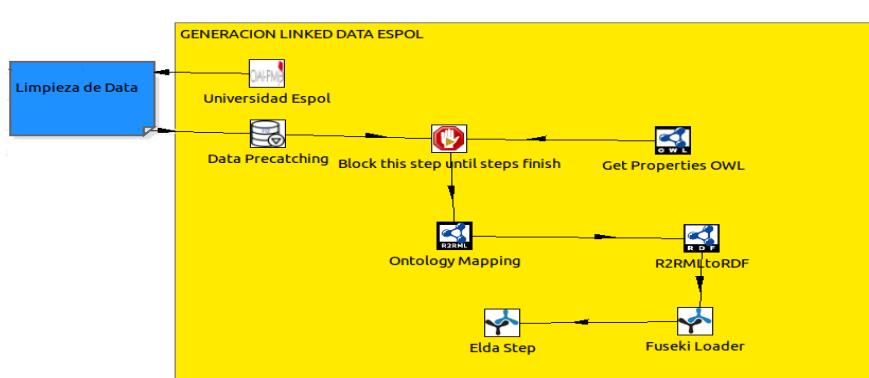


Figura 5.29: Configuración del Ejemplo práctico Sobre el repositorio ESPOL

Nota: El bloque azul de la imagen corresponde a un proceso de limpieza de datos realizado con los componentes propios del Pentaho Data Integration de acuerdo a la sección [5.2.2].

Capítulo 6

Conclusiones

Esta tesis empezó al observar la necesidad de poseer una plataforma que permita generar **Linked Open Data** (LOD) de una manera intuitiva, integrada y ágil, debido a que este proceso requiere un alto conocimiento de tecnologías semánticas para su utilización. A pesar de las investigaciones de generación de LOD, no existe una herramienta que permita abstraer la complejidad de cada etapa en este proceso, puesto que requiere la configuración de archivos complejos, además un alto tiempo de aprendizaje para su implementación. Teniendo en cuenta los problemas mencionados, en esta tesis se propone la creación de componentes para generar LOD, utilizando la plataforma Pentaho Data Integration (PDI). Se debe mencionar que todos los componentes desarrollados en esta tesis son compatibles entre sí y con los componentes propios de PDI.

A continuación se detallan los productos obtenidos:

1. Para la etapa de especificación se desarrolló el componente *OAILoader*, el cual solucionó el problema de extracción de los datos desde los repositorios OAIPMH.
2. Para la etapa de modelado de la metodología LOD se desarrolló el componente *Get Properties OWL* para Pentaho Data Integration, el cual permite cargar las propiedades y clases de cada ontología ingresada.
3. Para la etapa de generación se desarrolló el componente *R2RMLtoRDF*, el cual permite generar a partir de los datos en formato RDF, mediante una configuración en la interfaz gráfica del componente. Teniendo como entrada un archivo de mapeo y las fuentes de datos.

4. Para la etapa de publicación se desarrolló el componente *Fuseki Loader*, el cual genera una instancia del servidor Fuseki configurado automáticamente para que su triplestore cargue el RDF que se obtuvo en la etapa de generación. Además permite desplegar un servicio de SPARQL Endpoint que apunta al RDF ingresado para realizar su consulta.
5. Para la etapa de explotación se desarrolló el componente *Elda Step*, el cual permite generar una instancia de Elda configurada automáticamente. Al desplegar esta instancia el usuario tiene acceso a una interfaz amigable en la web que accede al servicio de SPARQL Endpoint generado en la etapa de explotación.

Como trabajos futuros se pretende, generar otros componente que agreguen nuevas funcionalidades tales como: la limpieza de errores en los datos en base a condiciones específicas de cada repositorio, generar instancias de otras tecnologías de publicación como Apache Marmota y Virtuoso, además sería de utilidad contar con un componente que facilite el proceso de enlace eligiendo una fuente de datos externa desde de PDI.

Capítulo 7

Anexos



UNIVERSIDAD DE CUENCA
desde 1867

7.1. Desarrollo de *plugins* para *Pentaho Data Integration*

Este anexo es una breve guía para la creación de nuevos componentes para la plataforma *Pentaho Data Integration* (PDI), aquí se detalla la estructura básica que tienen los componentes de PDI. Que al utilizar lenguaje de programación JAVA contiene clases, métodos y variables que un componente debe tener para su funcionamiento. Además se explica la manera que trabajan todas las clases en conjunto para conseguir el objetivo que propone el desarrollador. Finalmente el objetivo de conocer la estructura básica de los plugins, ayuda seguir las mismas directivas para el desarrollo de todos componentes y la facilidad de integración a PDI. la figura [7.1]. Muestra la interfaz de trabajo de PDI, en la cual se ejecutan todos los componentes.

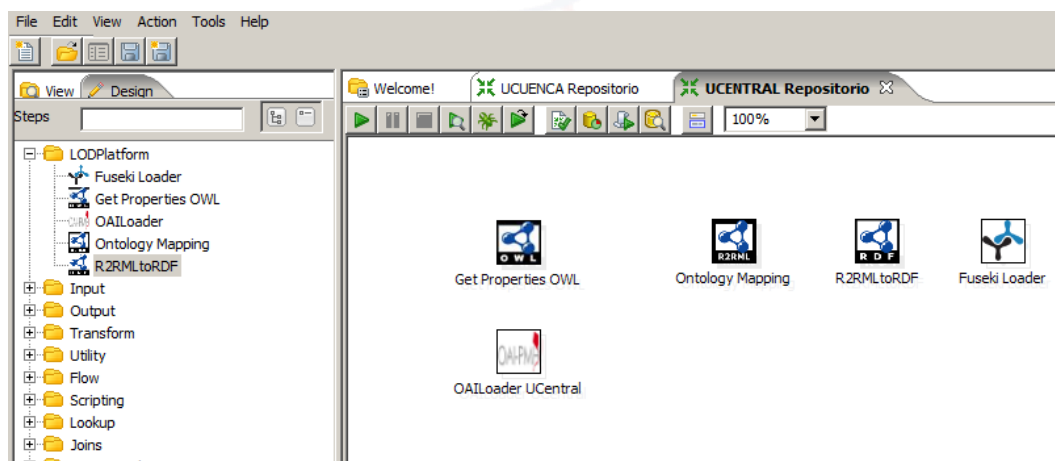


Figura 7.1: Interfaz gráfica de la plataforma de PDI

7.1.1. Estructura Basica de los *plugins* para *Pentaho Data Integration*

La estructura básica que cualquier componente en PDI se como ejemplo el componente OAILoader para la extracción de datos desde repositorios digitales OAI, del que se detalla su funcionalidad en el capítulo 3 y en el capítulo 4 se detalla su implementación. El cual es un componente creado siguiendo la metodología LOD para la publicación de *Linked Data* para este proyecto. Al crear un nuevo componente este es un proyecto JAVA *Maven* que consta de 4 clases, la figura [7.2]. describe el diagrama de clases con la estructura básica del componente

OAILoader.

- OAILoader.java
- OAILoaderData.java
- OAILoaderMeta.java
- OAILoaderDialog.java

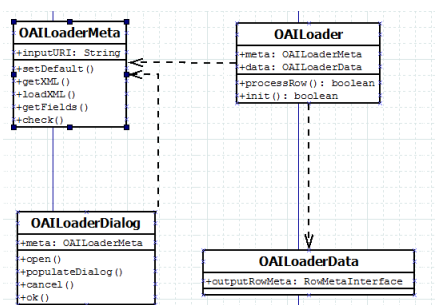


Figura 7.2: Diagrama UML de las clases de un componente de PDI

Además de las 4 clases los componentes manejan archivos para los mensajes para cada idioma, también una carpeta para sus recursos y al ser un proyecto *Maven* tienen el archivo POM. XML para agregar las dependencias que necesita el proyecto. En la figura [7.3] se ve el proyecto en eclipse del componente OAILoader.

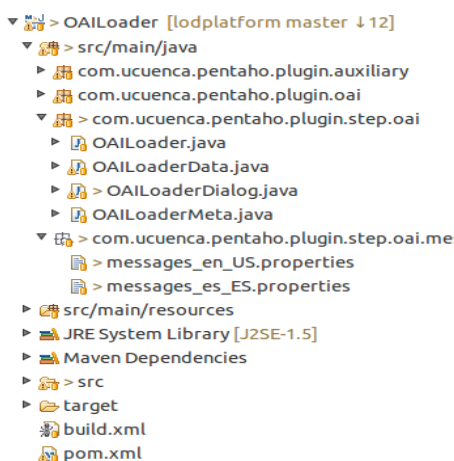


Figura 7.3: Proyecto eclipse del componente OAILoader

7.1.2. Análisis de las clases de un componente

En esta sección se analiza las clases de un componente con el objetivo de conocer cuál es su principal función, sus métodos y variables en cada una de ellas para conocer su aporte al componente.

7.1.2.1. Clase JAVA - OAILoader.java

La función principal de esta clase es permitir la comunicación entre la interfaz de PDI y el componente a través de un método existente en esta clase que permite el inicio de la ejecución del componente, esta clase es la principal de cada plugin.

7.1.2.2. Métodos de la clase OAILoader.java

Se explica los métodos más relevantes de esta clase.

processRow.- este método consta de dos parámetros que corresponde a los datos capturados por la interfaz del componente, los cuales son almacenados en el parámetro 'StepMetaInterface smi' que son enviados desde la clase OAILoaderMeta y a las variables de la clase OAILoaderData que le permiten guardar los datos de salida. ProcessRow es el método encargado del procesamiento de los datos de entrada al componente y colocar la salida del componente donde corresponda, que para el componente OAILoader es una tabla con los datos contenidos en un repositorio digital, pero puede existir diferentes salidas como archivos en diferentes formatos o ser la entrada para otro componente.

Declaración del método

```
public boolean processRow(StepMetaInterface smi, StepDataInterface sdi)
    throws KettleException {

    // safely cast the step settings (meta) and runtime info (data) to
    // specific implementations
    OAILoaderMeta meta = (OAILoaderMeta) smi;
    OAILoaderData data = (OAILoaderData) sdi;
    if(data.listRecords == null) throw new KettleException("ERROR WHILE RETRIEVING OAI RECORDS");
    if (first) {
        first = false;
        data.outputRowMeta = new RowMeta();
        meta.getFields(data.outputRowMeta, getStepname(), null, null, this);
    }
}
```

Figura 7.4: Declaración del método processRow

7.1.2.3. Clase JAVA - OAILoaderData.java

La función principal de esta clase es contener las variables que permiten el almacenamiento de los datos de salida en memoria.

Por lo general contiene la variable `public RowMetaInterface outputRowMeta`, pero se puede declarar mas variables según las necesidades en el desarrollo del componente.

7.1.2.4. Clase JAVA - `OAILoaderMeta.java`

La principal función de esta clase es la comunicación entre las clases `OAILoaderDialog.java` y `OAILoader.java`, también sirve para declarar las variables de configuración que se debe tener el componente en su funcionamiento.

7.1.2.5. Clase JAVA - Métodos

Se describirá los métodos más relevantes de la clase:

- `getXML`: Serializa el valor de una configuración guardada desde Dialog, por ejemplo el nombre ingresado para el Step.
- `loadXML`: Sirve para cargar el valor de una configuración guardada desde Dialog, por ejemplo el nombre ingresado para el Step.
- `getFields`: Determina los cambios que ha tenido un componente.
- `getDialog`: Este es un método para crea una instancia de la clase `OAILoaderDialog.java` que sirve para el intercambio de datos entre la clase `OAILoaderDialog` y `OAILoaderMeta`
- `setDefault`: Método que inicializa las variables, con los datos que aparecerán en el dialogo del componente.
- `check`: Sirve para verificar que todos los componentes obligatorios se hayan ingresado.

7.1.2.6. `OAILoaderDialog.java`

En esta clase se construye la interfaz del componente con el uso de la librería *Standard Widget Toolkit* (SWT). Por lo tanto al usar SWT los componentes son multiplataforma, además cada interfaz puede extender con los componentes necesarios hasta lograr la funcionalidad deseada.

PDI soporta componentes desarrollados en lenguaje de programación JAVA, por lo tanto permite la utilización de componentes ya liberados por la comunidad desarrolladora o la integración de nuevos componentes.

7.2. Configuración del modo de Depuración en *Pentaho Data Integration*

Es apartado trata de como configurar PDI en modo de depuración lo cual es una herramienta útil para el desarrollo de componentes. PDI esta configurado de manera predeterminada para ser usado como usuario final y no como desarrollador, por lo tanto resulta necesario configurar y activar la opción de depuración en PDI así como en IDE que en el caso actual fue Eclipse.

Se debe seleccionar un puerto que se asignara para la comunicaiion entre PDI y Eclipse, en este ejemplo es 8008. La primera parte de la configuración se realiza dentro de PDI, donde se debe editar el archivo Spoon.sh y descomentar las lineas que en la figura 7.5 se muestran.

```
# optional line for attaching a debugger
OPT="$OPT -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8008"
```

Figura 7.5: Configuración del archivo Spoon.sh

Una vez configurado el archivo, se debe configurar el IDE de desarrollo de Eclipse para que pueda obtener la ejecución del componente y realizar la depuración. La configuración que se debe realizar se muestra en figura 7.6.

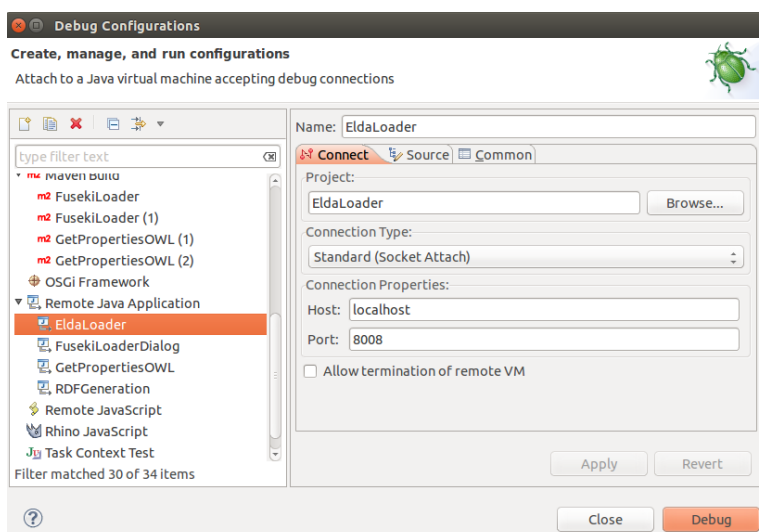


Figura 7.6: Configuración de Eclipse para la depuración de PDI.



Bibliografía

- [1] Antidot. Db2triples. <http://www.antidot.net/en/ecosystem/db2triples/>, 2005.
- [2] David Beckett. Rdf 1.1 turtle. 2014.
- [3] Oscar Corcho Asunción Gómez-Pérez Boris Villazón-Terrazas, Luis. M. Vilches. Methodological guidelines for publishing government linked data. *Information Systems Applications*, pages 27–49, oct 2011.
- [4] World Wide Web Consortium. *Guía Breve de Web Semántica*. <http://www.w3c.es/Divulgacion/GuiasBreves/WebSemantica>, 2013.
- [5] Pentaho Corporation. Pentaho data integration open source business intelligence. *Getting Started with Pentaho Data Integration*, pages 4–32, 2010.
- [6] Seema Cyganiak Richard Das, Souripriya Sundara. R2rml: Rdb to rdf mapping language. *W3C Recommendation*, 2012.
- [7] Carl Lagoze Herbert Van de Sompel.
- [8] APACHE SOFTWARE FOUNDATION. Getting started with apache jena, 2015. [Web; accedido el 1-02-2015].
- [9] Apache Software Foundation. Jena fuseki @ONLINE. *Phys. Rev. Lett.*, June 2015.
- [10] Jeff Heflin. An introduction to the owl web ontology language. *Lehigh University*.
- [11] Mercedes Marqués. *Bases de datos*. Publicacions de la Universitat Jaume I. Servei de Comunicació i Publicacions, Campus del Riu Sec. Edifici Rectorat i Serveis Centrals. 12071 Castelló de la Plana, 1993.

- [12] Thomas Mueller. H2 (dbms). 2011.
- [13] OAI-PMH. *Open Archives Initiative Protocol for Metadata Harvesting @ONLINE*. 2015.
- [14] Steve Harris Andy Seaborne. Sparql query language for rdf. 2013.
- [15] Deepti Gupta Neela G P Shrey Hatle, Ashiqa Sayeed. Pentaho data integration tool. *Business Intelligence Tool*, pages 2–18, 2013.
- [16] Epimorphics Linked Data Solutions. *Elda 1.3.5 (current) @ONLINE*. 2015.
- [17] Ed Summers. oai2rdf. <http://inkdroid.org/journal/2006/08/24/oai2rdf/>, 2006.
- [18] Ora Lassila Ralph R. Swick. Resource description framework (rdf) model and syntax specification. 08 October 1998.
- [19] Simón Mario Tenzer1. *Introducción a la Computación*. Archivos, formatos y extensiones, Septiembre 2007.
- [20] Lydia Chilton Dan Connolly Ruth Dhanaraj James Hollenbach Adam Lerer Tim Berners-Lee, Yuhsin Chen and David Sheets. Exploring and analyzing linked data on the semantic web. *Decentralized Information Group Computer Science and Artificial Intelligence Laboratory Massachusetts Institute of Technology Cambridge, MA, USA.*, November 6, 2006.
- [21] Belida Sudha Varanasi, Balaji. Introducing maven. *Computer Science*, pages 101–111, June 2014.
- [22] John Walkenbach. *Excel 2007 Bible*. 2007.